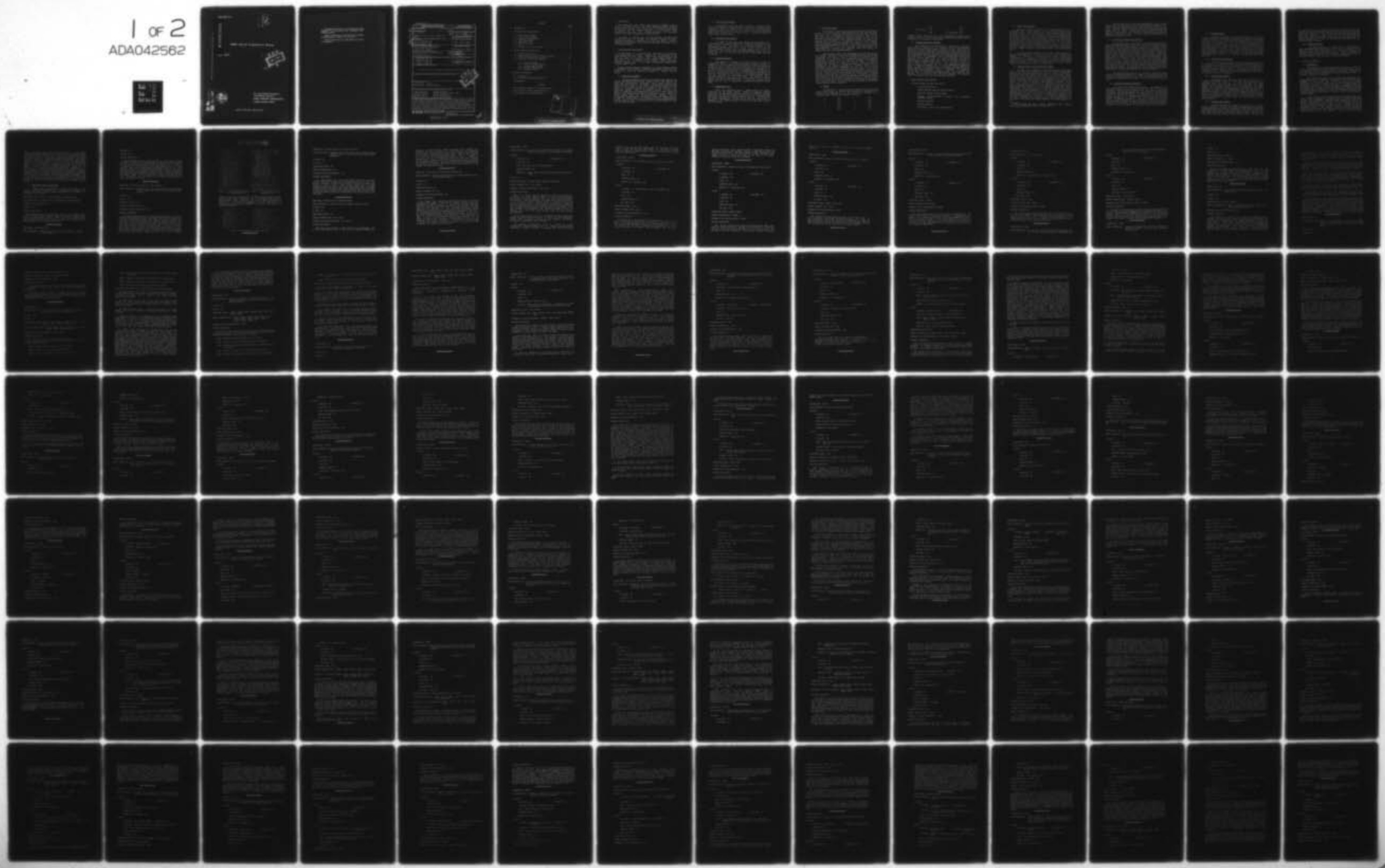
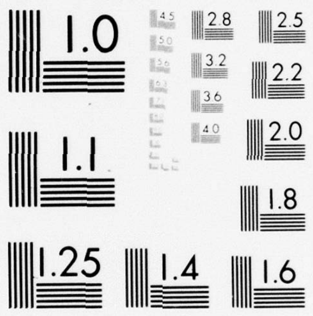
AD-A042 562 HARRY DIAMOND LABS ADELPHI MD F/G 9/2
ARMS REMOTE PROGRAMMER'S MANUAL.(U)
JUL 77 D J BÜSCHER, M A RESSLER
UNCLASSIFIED HDL-SR-77-1 NL

1 OF 2
ADA042562




1.0
4.5
5.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

HDL-SR-77-1

12 NW

SR-77-1—*ARMS Remote Programmer's Manual,* by David J. Buscher and Marc A. Ressler

ARMS Remote Programmer's Manual

July 1977

U.S. Army Materiel Development and Readiness Command

HARRY DIAMOND LABORATORIES

Adelphi, Maryland 20783

RESEARCH DEVELOPMENT ENGINEERING

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. ~~REPORT NUMBER~~ (14) HDL-SR-77-1	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) ARMS Remote Programmer's Manual,		5. ~~TYPE OF REPORT & PERIOD COVERED~~ (9) Special Report,
		6. PERFORMING ORG. REPORT NUMBER
7. ~~AUTHOR(s)~~ (10) David J. Buscher, Marc A. Ressler		8. CONTRACT OR GRANT NUMBER(s) PRON: WD6 61 WDA9
9. ~~PERFORMING ORGANIZATION NAME~~ AND ADDRESS Harry Diamond Laboratories, 2800 Powder Mill Road, Adelphi, MD 20783		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Ele: MDINT
11. CONTROLLING OFFICE NAME AND ADDRESS Bonneville Power Administration, P.O. Box 3621, Portland, OR 97208		12. ~~REPORT DATE~~ (11) July 1977
		13. ~~NUMBER OF PAGES~~ 125
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) (12) 121p.		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

HDL Project: 543652
DRCMS Code: 697000.00.00000

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Power metering	Energy metering
Revenue metering	Automatic meter reading (AMR)
Demand metering	Reactive metering
Microprocessor	Solid state metering

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A general description of the Automatic Revenue Metering System (ARMS) Remote Metering Device is given. This description includes the functions of the various system hardware devices and their associated software modules. There is also detailed documentation for each subroutine of the ARMS Remote software. This software includes a real-time interrupt-driven operating system and all the necessary utility functions for performing revenue metering and the associated reporting functions.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

163050

[illegible]

CONTENTS

Page

1. INTRODUCTION 5

2. SYSTEM HARDWARE CONFIGURATION 5

2.1 Meter Input Hardware 5
2.2 Form-C Output Hardware 6
2.3 Real-Time Clock Hardware 6
2.4 Telemetry Hardware 6
2.5 Demand Tape Drive 6
2.6 Self-Scan Display 7
2.7 Keypad 7
2.8 Teletype and RS-232 Interface 8

3. SYSTEM SOFTWARE CONFIGURATION 8

3.1 System Initialization 9
3.2 Revenue Metering Input and Output Programs 9
3.3 Real-Time Software-Clock Processor 10
3.4 Interrupt Processor and Handlers 10

3.4.1 Teletype Handler 11
3.4.2 Self-Scan Display Handler 11
3.4.3 Console Keypad Handler 11
3.4.4 Cartridge Tape Handler 11
3.4.5 Telemetry Handler 12

4. PROGRAM DOCUMENTATION 12

4.1 Introduction 12
4.2 Individual Program Documentation 13

DISTRIBUTION 125

FIGURES

1 Data structure of header for input-data block 15

2 Data structure of meter input-data block 15

3 Data structure of output-data block 15

1. INTRODUCTION

The information in this manual should allow a programmer to become familiar with and to maintain the ARMS (Automatic Revenue Metering System) Remote Software. It is assumed that the reader of this manual is familiar with the assembly language of the Motorola M6800 microprocessor and also has a working knowledge of the functional aspects of the M6800 microprocessor family chip set.

The manual is divided into four main sections. The first three sections deal with the software and hardware in general and contain information necessary to understand the rest of the manual. Section 4 summarizes documentation on the individual programs that make up the ARMS software system.

2. SYSTEM HARDWARE CONFIGURATION

The ARMS Remote is a microprocessor system based on the Motorola M6800 Microprocessor Unit (MPU). Programs for the operation of the system reside in read only memory (ROM); dynamic system parameters and data are stored in read/write random access memory (RAM). All input/output (I/O) is handled by the Motorola MC6820 Peripheral Interface Adapter (PIA) or the MC6850 Asynchronous Communications Interface Adapter (ACIA).

Besides the microprocessor components, the system contains a number of peripheral devices necessary to perform the metering functions of the system and to allow the sub-station operator to interact with the system.

2.1 Meter Input Hardware

The main peripheral devices are the kilowatt-hour (KWH) and the kiloVAR (volt-ampere-reactive) hour (KVARH) meters (both meters are solid-state watt/watt-hour transducers). These meters present both Form-C and analog inputs to the system. The system provides a wetting voltage to the K contact of the Form-C, and the Y and Z contacts of the Form-C are connected through an interface to two bits of a PIA data-input register. Thus it is possible to detect a Form-C state change by examining the corresponding PIA bits for a state change on each bit. The analog meter outputs are connected to either a KW or KVAR analog summing input. An analog-to-digital converter (ADC) is periodically switched between the outputs of these two summing circuits to read the instantaneous KW or KVAR values for all connected meters. The 8-bit digital value is transferred into the system through an 8-bit PIA input data register.

2.2 Form-C Output Hardware

The system can also use Form-C relays as outputs to drive conventional totalizers, demand meters, or other devices requiring relay closures for inputs. The output Form-C relays are controlled by PIA output-data registers with one bit of the PIA controlling one associated Form-C relay. When the PIA bit changes state, it causes the connected Form-C relay to change state.

2.3 Real-Time Clock Hardware

In order to keep the time of day, the system relies on a crystal-controlled 1-MHz oscillator and a divider which supplies a 10-ms clock period. This 10-ms clock is connected directly to the non-maskable interrupt (NMI) line of the MPU. Therefore every 10 ms the clock is updated for 10-ms units, seconds, minutes, hours, and days as appropriate. This software clock times demand periods, keeps the time of day and date, and performs other functions which require real-time information.

2.4 Telemetry Hardware

The Remote is connected to a Master Station by a two-way communication network. The communication system to Remote interface is handled by a set of telemetry tone transmitter and receiver cards which transmit and receive serial data from a microwave link. The digital I/O to these cards is serial data which is generated or accepted by the serial I/O ports of an ACIA chip. All communication is synchronous by bit and asynchronous by 8-bit bytes. The data byte is preceded by one start bit and followed by two stop bits. The telemetry channel rate is 110 baud, operating in a full-duplex mode. The telemetry channel allows the Remote to send meter and other system data to the Master Station and allows the Master Station to query and control certain Remote functions such as generating a system-wide synchronous freeze pulse.

2.5 Demand Tape Drive

Meter data generated during a demand period are usually transmitted to the Master Station. However, if telemetry problems exist, then this data must be captured locally on the Remote's cartridge tape drive. The tape drive accepts 8 bits of data in a parallel format and records these data in a serial format on the tape. The capacity of one tape is approximately 100,000 bytes, with the actual density depending on the length and number of inter-record gaps, and the ratio of ones to zeros.

2.6 Self-Scan Display

All messages and requests for data are handled by the self-scan display. It is a 16-position alphanumeric display with character entry from right to left as observed from the front of the display. Although there are only 16 displayable character positions, there is an additional 16-character buffer which stores characters as they move off the left end of the display. This additional buffer is used in the backspace mode to recall up to 16 characters that have been "flushed" from the visible area of the display. For example, if the message "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345" was output to the display with A the first character and 5 the last, then the visible message would be "QRSTUVWXYZ012345." However, if 6 backspaces were then output, the message would then read "KLMNOPQRSTUVWXYZ."

The six display data bits correspond to the lower six bits of the standard seven-bit ASCII (American Standard Code for Information Interchange) code table. These bit positions are connected to PB0 through PB5 on a PIA chip. PB6 is connected to the display backspace control and is normally high to indicate no backspace. To initiate a backspace this bit must be pulsed low for at least 1 μs and then returned high. The end of a backspace sequence or character output is indicated by the display generating a low-going pulse on the "data-taken" line, which is connected to the CB1 line of the display PIA. PB7 is connected to the display's blanking control line. A low state on this line is normal and allows the characters on the display to be seen. A high state on this line causes the display screen to become dark, but no data are lost. Data are output to the display by putting the ASCII data in the PIA data register, making sure PB7 and PB6 are the right sense. The PIA CB2 control is then activated to cause a low-going pulse on the CB2 line, which is connected to the display's data present line. When the display is ready for the next character, the data-taken line connected to CB1 is pulsed low.

2.7 Keypad

The keypad is a 16-key solid state keyboard which is connected to the A and B sides of one PIA. Each key output is connected to one input on the attached PIA and all key closures are indicated by a low-going pulse. The keys are connected as follows.

1	PB0	.	PB5
2	PB1	0	PB6
3	PB2	*	PB7
;	PB3	7	PA0
,	PB4	8	PA1

9	PA2	5	PA5
# ("execute")	PA3	6	PA6
4	PA4	$ ("attention")	PA7

A second output from each key is connected through a common pull-up resistor to the PIA CB1 control line. Therefore, any time any key is depressed, an IRQ interrupt is generated.

2.8 Teletype and RS-232 Interface

The Teletype or an RS-232 serial input device (simulated Master Station) is connected to the system by a serial interface which generates the proper signal levels to attach to an ACIA serial input or output line. Either a Teletype or RS-232 device can be connected to the proper rear-panel connector and proper operation is assured by selecting the proper baud rate (by a switch on the interface board) to match the connected device. The programmer need be familiar only with the ACIA since its interface serves only as a signal conditioner. To output a character to a serial device, the ACIA is primed according to Motorola-defined procedures with CR0 = 1, CR1 = 0, CR2 = 0, CR3 = 0, CR4 = 1, CR5 = 1, CR6 = 0, and CR7 = 0. One character is output each time an interrupt is received and the ACIA transmit-data register is empty. To clear the interrupt flag, a read-data register instruction must be executed. To prepare to receive a character, CR0 through CR4 are the same as above and CR5 = 0, CR6 = 0, and CR7 = 1. As soon as a key is hit, an interrupt is generated and the "ACIA Receiver Data Register Full" flag will be set. The act of inputting the character resets the interrupt flag bit and processing continues.

3. SYSTEM SOFTWARE CONFIGURATION

The ARMS software has seven main parts:

System initialization

Revenue metering input and output programs

Real-time software clock programs

External device-interrupt processor and associated interrupt handlers

Wait-list processor

Message processor

FIFO (first in/first out) stack processor

3.1 System Initialization

There are certain volatile system parameters that must be initialized when an ARMS Remote Unit is first installed. These parameters include pulse counts, and other meter-related data pointers for tables, stacks, and system constants. Some of these parameters can be initialized by the system without outside interaction because their values are well defined. Other parameters, such as the scale factors for specific inputs, must come from external sources. In the prototype this information is acquired through interaction with the user either through the Teletype or Console Keypad. There is also a set of meter information stored in ROM. These data are for a predefined site and can be used for system checkout. In subsequent systems, this initialization would be handled directly from the Master Station or by preprogrammed ROM's.

The initialization procedure, at this time, takes the form of a question and answer session with the system user. This initialization dialogue is defined in the ARMS Remote User's Manual.[1] Once initialization is completed and checked, the system starts normal operation as soon as the trigger signal for the start of the next demand period is received. This signal can be from an external Form-A contact closure or synchronized to start at a preset time.

3.2 Revenue Metering Input and Output Programs

The heart of the software is programs RIN and ROT which handle Form-C inputs and outputs respectively. The RIN program sequentially examines the PIA registers associated with the Form-C meter inputs. An eight-bit PIA data register can handle up to four Form-C inputs. Each pair of bits serves as an input for two contacts of a Form-C meter input. The initial state of each PIA is maintained in the system and when two paired bits change state, a Form-C "pulse" is recorded by the system. The state of the PIA is then updated to reflect the Form-C pulse. Once a pulse is recognized, all appropriate data for the pulsed input are updated accordingly. If any outputs are "connected" to the pulsed input, then the data for the associated outputs are updated accordingly. If this update causes a request for a Form-C output pulse, then the PIA bit (which corresponds to the specified output relay) is complemented, which causes the relay to change state. Once all input PIA's have been checked, the FIFO list is examined to determine if any programs are waiting to be executed. If so, the first program of the FIFO list is executed, then control returns to RIN for another check of all inputs. The nominal time between execution of the RIN program is 10 ms.

---

[1]*David J. Buscher and Marc A. Ressler, ARMS Remote User's Manual, Harry Diamond Laboratories SR-76-3* (December 1976).

The other portion of the metering measurements is the periodic sampling of the analog output of both the KW and KVAR transducers. This analog signal is proportional to the instantaneous KW or KVAR readings of all connected analog transducer inputs. All KW analog signals are summed, and the signal is converted to digital format by an ADC. The same ADC is then switched to the sum for all KVAR analog signals and the signal is converted and input. The ADC switches between KW and KVAR readings at a nominal 6-s rate.

3.3 Real-Time Software-Clock Processor

The software clock is equivalent to the conventional mechanical clock which times the demand period and generates a freeze signal to indicate the end of the demand period. The key to the software clock is an external oscillator and a frequency divider which generate a pulse every 10 ms. This pulse drives the NMI line of the M6800 microprocessor and causes the NMI handler to be executed. This routine, called CLK, updates pulse, second, minute, hour, and day counters as necessary. These counters, except for pulse, are all in binary coded decimal (BCD) format and are maintained in the RAM portion of system memory. Also maintained are the second, minute, and hour when the current demand period will end. One pulse time (10 ms) before the end of the demand period, a prefreeze program is initiated to lock out extraneous programs from execution and to bring the current demand-period data collection to an orderly termination. The next NMI interrupt indicates the end of the demand period, updates all appropriate pulse counters and transfers meter data to the local demand tape and to the Master Station.

The clock processor also has an auxiliary role which governs how often the wait-list processor is called. The Check Wait List (CWL) routine is called periodically by the clock processor and the wait list is checked to determine if it is time to recall a program from the wait list.

3.4 Interrupt Processor and Handlers

All interrupts, except NMI interrupts, are handled by the interrupt processor. If the interrupt mask bit (Im) of the microprocessor is zero and the IRQ line is active (low) then the IRQ program is executed. This program examines all status registers for ACIA's and PIA's whose addresses are found in the IRQPNT table. If the interrupt flag bit is set in any of these status registers then the associated handler (determined by a further entry in the IRQPNT table) is given control. If extensive processing is required to handle the interrupt, then the extensive handler is placed on the wait list or FIFO stack for execution at a later time. If the interrupt can be handled quickly it is done by the initial handler and then an RTI instruction is executed.

3.4.1 Teletype Handler

The Teletype Handler is executed whenever a key on the Teletype keyboard is depressed and (when allowed) whenever the Teletype is ready to accept a character from the microprocessor. For Teletype input, the key character is input and placed in a buffer. If the character is a carriage return, then the message processor (MSPRO) is put on the FIFO stack to await later execution. If the character is not a carriage return, then an RTI instruction is executed. On the output side, characters are output from a buffer one at a time with an RTI after each output. Then when a zero byte is encountered in the output buffer, the output sequence is terminated. At this time the program which requested the output is put on the FIFO stack and awaits later execution.

3.4.2 Self-Scan Display Handler

The Self-Scan Display Handler uses its PIA-interrupt flag bit for operation, but it does not affect the IRQ interrupt and thus it is not entered via the IRQ-interrupter processor. It is a callable routine with several entry points to display a message, clear the display, or execute a number of backspace commands. The nominal time between output characters is 66 μs; therefore, it takes approximately 1 ms to output a 16 character message to the display.

3.4.3 Console Keypad Handler

The keypad has 16 output lines, each of which corresponds to a specific key closure. These 16 lines are used as inputs to the 16 input-data lines of a full PIA. One other line from the keypad generates an IRQ interrupt any time a key is depressed. The Keypad Handler, which receives control from IRQ, the interrupt processor, examines the 16 PIA data lines sequentially. The first key found to be depressed is converted to its corresponding ASCII code and saved in a special buffer area. (If more than one key is depressed, only the first key in the examination sequence will be found.) The handler then returns to the interrupted program via an RTI instruction, or, under certain circumstances, a special program is put on the FIFO stack to await execution.

3.4.4 Cartridge Tape Handler

The Cartridge Tape Unit uses one full PIA for data, status checks, and control. The tape unit handler is characterized by critical timing constraints which must be observed to insure proper reading from and writing to the tape. To insure that these constraints are observed, the system "locks out" all noncritical functions during tape read and write times. Metering functions and the real-time clock continue, but all other functions are temporarily suspended. The A-side of the PIA contains all status and control bits for the tape drive. This includes tape ready, beginning of tape, buffer ready, and incomplete character, which are status bits. The control bits are run pulse, read line enable, tape direction, read/write control, stop pulse, and reset. The B-side contains the data bits for the drive and inter-record gap, and load write buffer controls.

3.4.5 Telemetry Handler

The Telemetry Handler is responsible for all messages and data transferred between the Master Station and the Remote. In the prototype Remote this handler serves as a local link between the Teletype input and output functions. All data traffic to and from the Teletype is passed through the local Telemetry transmitter/receiver or receiver/transmitter loop and then to or from the Teletype.

4. PROGRAM DOCUMENTATION

4.1 Introduction

The documentation for each program has two parts. The first is a brief program description which details the program function, how it is entered, how it exits, and how it alters the system registers and RAM. The second part of the documentation gives a detailed account of the inner workings of the program.

All program names are up to six alphanumeric characters long. Program labels within a program usually use the first three or four characters of the program name followed by one or two numeric characters. This rule is violated only if a more meaningful label name is required for clarity. Variables used within a program follow the same rule as labels, except that the two numeric characters are replaced by an alphabetic character (sometimes followed by a number, but rarely by another alphabetic character). The first alphabetic character is an X for double-byte storage locations and an A, B, or C for a single-byte storage location.

There are three other conventions that the reader should become familiar with to understand the programs clearly. The first deals with so-called "IRQ" interrupts. These interrupts are maskable by the Im bit present in the condition code register. If this bit is set, either by an SEI instruction or as a result of an interrupt, then no IRQ interrupts will be recognized by the processor. Except for special cases where an SEI . . . CLI sequence is used to disable IRQ interrupts during critical processing, all CLI functions after an IRQ interrupt are performed as part of the RTI sequence. Since the Im bit is cleared before an interrupt sequence, the condition code with a clear Im bit will be pushed onto the stack with a clear Im bit. Then during the RTI sequence, this same condition code will be restored, thereby clearing the interrupt mask bit and allowing further IRQ interrupts. The second convention deals with determining program address-list terminators. Often it is desired to determine the end of a set of two-byte program addresses of a sequential list. One way is to set two consecutive bytes equal to zero and therefore terminate the list when a 16-bit zero address is found. However, this method can be made simpler when we realize that no programs start below location OOFF. Therefore, it is only necessary to have a high-order zero byte as a terminator address. This convention is important and is used quite frequently. The third convention deals with calling subroutines by a JSR or BSR instruction. When the last two instructions of that called routine are normally a JSR or BSR followed by an RTS, this is shortened to either a JMP or BRA instruction to the called routine. Therefore, when the second called routine does an RTS, it will return to the first caller rather than the second.

4.2 Individual Program Documentation

Programs appear in the order in which they appear in the listing. For proper understanding it is necessary to examine the listing (available from Harry Diamond Laboratories) while reading the description of a program.

TABLE NAME - Card, Chip, and Register Address Definition Tables

BRIEF DESCRIPTION - Symbolic names are assigned to hardware devices.

ON ENTRY - Not Applicable (NA)

ON EXIT - NA

DETAILED DESCRIPTION -

These tables relate physical hardware devices such as memory chips and PIA and ACIA chips to specific addresses within the ARMS software system. If certain device addresses change, then their addresses need only be changed in these tables since the devices are referenced by the table symbolic names in the software proper.

TABLE NAME - System Parameters

BRIEF DESCRIPTION - Symbolic names and values are assigned to certain system parameters.

ON ENTRY - NA

ON EXIT - NA

DETAILED DESCRIPTION -

System parameters are values used throughout the software as system constants. If the system is reconfigured to have fewer IRQ interrupts, then the value NINT is updated along with the IRQPNT table. If more RAM is needed, the initial value of the stack pointer (SP) is simply changed. The symbol IIDHSZ defines the number of bytes in an input data-block header. If the requirements change for the data structure of the header then this value must be changed. SYN, GO, and TRG are ASCII values for the startup procedure (see routine STR). NIDTA and NODTA are the maximum numbers for the "WHICH DATA ??" query of routines OP00 and OP01. These values must be changed if the number of data queries in these two programs is changed.

TABLE NAME - Relative Pointers for Input Data Block

BRIEF DESCRIPTION - Symbolic names, with values equal to their relative position within the data block, are assigned to data pointers.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - NA

DYNAMIC VARIABLES USED - NA

VARIABLES TO BE PRE-INITIALIZED - NA

DETAILED DESCRIPTION -

Each Form-C relay input has an associated data-block area. Since an 8-bit PIA peripheral-data register can handle four Form-C inputs the data blocks are grouped in fours. The header record for a group of four is shown in figure 1 with the relative pointers RRADDR, CVAL, RR1, and NEXT shown at their respective locations. Each header contains four double-byte addresses starting at RR1 which point to the four associated input data blocks. The address at NEXT points to the address of the next header if there is one. If not, the NEXT address is all zeros. The relative pointers PWRDIR through MAXDEM point to their respective relative locations within an input data block (see fig. 2). OUTPNT

PIA ADDRESS (HIGH)	(RRADDR)
PIA ADDRESS (LOW)	
CURRENT VALUE OF PIA INPUT	(CVAL)
ADDR. OF 1ST DATA AREA FOR SPECIFIED PIA (HIGH)	(RR1)
ADDR. OF 1ST DATA AREA FOR SPECIFIED PIA (LOW)	
ADDR. OF 2ND (HIGH)	
ADDR. OF 2ND (LOW)	
ADDR. OF 3RD (HIGH)	
ADDR. OF 3RD (LOW)	
ADDR. OF 4TH (HIGH)	
ADDR. OF 4TH (LOW)	
ADDRESS OF NEXT HEADER BLOCK (HIGH)	(NEXT)
ADDRESS OF NEXT HEADER BLOCK (LOW)	

NOTE-EACH HORIZONTAL BOX EQUALS 1 BYTE.

Figure 1. Data structure of header for input-data block.

POWER TYPE 1=KWH 2=KVARH	POWER FLOW 1=IN 2=OUT	(PWRTYP AND PWRDIR)
NUMBER OF PULSES IN CURRENT PERIOD (HIGH)		(NPULC)
NUMBER OF PULSES IN CURRENT PERIOD (LOW)		
NUMBER OF PULSES IN LAST PERIOD (HIGH)		(NPULL)
NUMBER OF PULSED IN LAST PERIOD (LOW)		
NUMBER OF PULSES IN ALL PERIODS EXCEPT CUR. (HIGH)		(NPULT)
NUMBER OF PULSES IN ALL PERIODS EXCEPT CUR. (MED.)		
NUMBER OF PULSES IN ALL PERIODS EXCEPT CUR. (LOW)		
SCALE FACTOR (HIGH)		(INSCL)
SCALE FACTOR (LOW)		
LAST PULSE TIME		(LPULT)
MAXIMUM PULSE TIME		(MPULT)
MAXIMUM DEMAND VALUE (HIGH)		(MAXDEM)
MAXIMUM DEMAND VALUE (LOW)		
OUTPUT REFERENCES (LIST)		(OUTPNT)

Figure 2. Data structure of meter input-data block.

points to the first location used to indicate which outputs are associated with the input. OUTPNT is a list of variable length containing binary numbers which are equal to the output numbers referenced by the input. A zero byte is a terminator. If the terminator is at OUTPNT then no outputs are referenced. See figure 3 for details of the block structure.

PIA ADDRESS (HIGH)	(OUTPIA)
PIA ADDRESS (LOW)	
POWER TYPE	(PTYPE)
NUMBER OF PULSES IN CURRENT PERIOD (HIGH)	(NOUTC)
NUMBER OF PULSES IN CURRENT PERIOD (LOW)	
NUMBER OF PULSES IN LAST PERIOD (HIGH)	(NOUTL)
NUMBER OF PULSES IN LAST PERIOD (LOW)	
NUMBER OF PULSES IN ALL PERIODS EXCEPT CUR. (HIGH)	(NOUTT)
NUMBER OF PULSES IN ALL PERIODS EXCEPT CUR. (MED.)	
NUMBER OF PULSES IN ALL PERIODS EXCEPT CUR. (LOW)	
SCALE FACTOR (HIGH)	(OUTSCL)
SCALE FACTOR (LOW)	
FRACTIONAL PULSE CNT.(HIGH)	(SOUTC)
FRACTIONAL PULSE CNT. (LOW)	
PIA OUTPUT MASK	(OUTMSK)

Figure 3. Data structure of output-data block.

TABLE NAME - Relative Pointers for Output Data Block

BRIEF DESCRIPTION - Symbolic names, with values equal to their relative position within the data block, are assigned to data pointers.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - NA

DYNAMIC VARIABLES USED - NA

VARIABLES TO BE PRE-INITIALIZED - NA

DETAILED DESCRIPTION -

The output blocks are all of fixed length and have no header records. The OUTPIA location points to the PIA register to which the output is attached. If there is no attachment to a physical device this location is set to zeros. PTYPE through OUTMSK point to their respective relative locations within the output-data block. The OUTMSK location has all bits set to zero except for those bits which reference an external device attached to the associated PIA. Each bit set in this location will cause a relay state change in the event of an output pulse requirement.

TABLE NAME - Storage Area for System Variables

BRIEF DESCRIPTION - Locations where all dynamic values are stored.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - NA

DYNAMIC VARIABLES USED - See listing

VARIABLES TO BE PRE-INITIALIZED - See listing

DETAILED DESCRIPTION -

This area is for storage of system values that are changeable. The locations referenced are all in RAM and are therefore readable and writable. Except for certain global variables, most variables are referenced to a specific program by the first three or four characters of their names. In some cases, certain RAM locations may serve more than one master through the equivalence table. Some of the values in this area must be initialized by the system. This is handled by programs as part of the initialization phase. This area also contains several tables of buffers as well as the system stack used for interrupt and subroutine processing. The uses and structures of these tables and buffers will be described in more detail during the documentation of the programs that make use of them.

TABLE NAME - Fixed System Constants and Character Data

BRIEF DESCRIPTION - Contains ASCII data for all system messages and certain system constants.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - NA

DYNAMIC VARIABLES USED - NA

VARIABLES TO BE PRE-INITIALIZED - NA

DETAILED DESCRIPTION -

This storage area contains all message and individual character constants of the system. It contains the OP-CODE look-up table with associated jump address table. It also contains all data and tables necessary to perform a pseudo-master-station initialization. The ADDRm values are the addresses of the individual data-entry blocks. Computation of these values must take into account the variable lengths of the input data block. The IHEADm addresses correspond to the tops of all input-data-block headers. The IDTAm addresses point to the areas where the pseudo-master-station input-data-block filler information is. There is a one to one correspondence between these addresses and the number of inputs into the system. The ODTAm addresses point to the output-data-block filler information. NIN and NOUT are the number of inputs and outputs, and KWFAC and KVAFAC are the values of ADC increments for KW inputs and KVAR inputs.

PROGRAM NAME - START

BRIEF DESCRIPTION - Initializes all ACIA and PIA registers and certain dynamic RAM variables. Also the tape is initialized.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Stack pointer initialized by START

INTERRUPT STATUS - NA

ENTERED VIA - Reset switch activation through Interrupt Pointer Table

ON EXIT - NA

SUBROUTINES CALLED - CLRMEM, CHKTAP, REWIND, LOADP, SSP16

DYNAMIC VARIABLES USED - See listing

VARIABLES TO BE PRE-INITIALIZED - See listing

DETAILED DESCRIPTION -

This is the routine executed whenever the system reset function is activated. Its starting address appears in the reset and interrupt pointer table at location FFFE and FFFF. This routine first initializes the system SP. Since non-maskable interrupts (NMI) happen every 10-ms, it is necessary to have a defined stack to handle the interrupt initiation and restoration. If an interrupt happens after the reset and before the execution of the LDS #SP instruction, havoc will ensue and it will be readily apparent that the reset should be pressed again. It is also necessary to immediately zero out the INIT flags so that the software clock and the input routines will not start running uninitialized.

The run time constants are then initialized by first clearing the memory (since many locations need to be zero), and then initializing certain other locations by a series of load immediate instructions followed by an appropriate store instruction.

This parameter initialization phase is followed by a tape initialization phase which makes use of a number of tape-oriented subroutines. The tape is rewound if necessary (BSR REWIND) then moved forward to the load point (BSR LOADP) and then checked to see if the tape is ready for recording (BSR CHKTAP). If the tape is ready, execution proceeds to location STA3. If not, the system loops until the tape is loaded correctly.

PROGRAM NAME - CHKTAP

BRIEF DESCRIPTION - Checks whether the tape is ready for recording.

ON ENTRY -

A REGISTER - NA	B REGISTER - NA
X RESISTER - NA	
STACK - NA	
INTERRUPT STATUS - NA	
ENTERED VIA - Subroutine call	

ON EXIT -

A REGISTER - 0 = Ready, Nonzero = Busy	B REGISTER - NA
X REGISTER - NA	
STACK - NA	
INTERRUPT STATUS - NA	
EXIT MODE - RTS	

SUBROUTINES CALLED - SSP16

DYNAMIC VARIABLES USED - CHKTA

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine puts the tape drive in record mode and resets the tape drive. If the tape cartridge does not have the write-enable set or there is some other malfunction, then the tape-ready bit (PA7) will be "1"; therefore, the BEQ CHKT1 instruction will be passed and the display routine will display the message "TAPE NOT READY." Also CHKTAP will return to the caller with a non-zero value. If the tape is ready, the BEQ CHKT1 instruction will transfer control to CHKT1 and return to the caller with a zero value. The variable CHKTA is used to assure that the display routine is called only once if the tape is not ready. Otherwise, the display would not be readable.

PROGRAM NAME - REWIND

BRIEF DESCRIPTION - Rewinds the tape to the beginning of tape point.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - RUNTAP, SSWAIT

DYNAMIC VARIABLES USED - TAPBSY

VARIABLES TO BE PRE-INITIALIZED - TAPBSY

DETAILED DESCRIPTION -

This routine examines the tape-drive beginning-of-tape (BOT) flag (PA6) and branches to REW2 if the tape is at the BOT point. Otherwise, the tape is set to reverse and read mode, and a run pulse is given. The BOT flag is then read periodically and when BOT is reached the program exits after clearing the TAPBSY flag which was set by the RUNTAP routine.

PROGRAM NAME - LOADP

BRIEF DESCRIPTION - Moves a tape from the BOT point to the load point.

ON ENTRY -

A REGISTER - NA	B REGISTER - NA
X REGISTER - NA	
STACK - NA	
INTERRUPT STATUS - NA	
ENTERED VIA - Subroutine call	

ON EXIT -

A REGISTER - NA	B REGISTER - NA
X REGISTER - NA	
STACK - NA	
INTERRUPT STATUS - NA	
EXIT MODE - RTS	

SUBROUTINES CALLED - RUNTAP, STOP, WAIT

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

The load-point routine assumes the tape is at the BOT point. It places the drive in the forward read mode and issues a run pulse. It then periodically checks the status of the tape-ready bit. When the ready bit goes low, the BLT LOA1 instruction will drop through, the tape will be stopped, and control returned to the calling program.

PROGRAM NAME - RUNTAP

BRIEF DESCRIPTION - Pulses the tape drive run bit which causes tape motion to start in preselected direction

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - TAPWRK

DYNAMIC VARIABLES USED - TAPBSY

VARIABLES TO BE PRE-INITIALIZED - TAPBSY

DETAILED DESCRIPTION -

This routine first checks to see if the tape is already busy. The TAPBSY flag is examined, and if busy (non-zero), the TAPWRK routine is called. This routine will maintain control until TAPBSY goes to zero. Control then returns to RUN0. At this point TAPBSY is set non-zero to show that it is busy, and the run bit of the tape drive is set low for one instruction time (STAA PIATP1) and then brought high by the following instruction (STAB PIATP1).

PROGRAM NAME - STOP

BRIEF DESCRIPTION - Stops tape motion.

ON ENTRY -

A REGISTER - NA	B REGISTER - NA
X REGISTER - NA	
STACK - NA	
INTERRUPT STATUS - NA	
ENTERED VIA - Subroutine call to STOP or STO1	

ON EXIT -

A REGISTER - NA	B REGISTER - NA
X REGISTER - NA	
STACK - NA	
INTERRUPT STATUS - NA	
EXIT MODE - RTS	

SUBROUTINES CALLED - WAIT

DYNAMIC VARIABLES USED - TAPBSY

VARIABLES TO BE PRE-INITIALIZED - TAPBSY

DETAILED DESCRIPTION -

This routine pulses the tape-drive-stop bit in a manner similar to the run pulse of RUNTAP. After the stop bit has been pulsed it takes time for the tape to stop. Therefore, the routine waits for 20 ms before returning control to the caller. Then the TAPBSY flag is set non-busy and control is returned to the caller.

PROGRAM NAME - TAPWRK

BRIEF DESCRIPTION - If the tape is busy and requested by another user this program displays the message "**WORKING**" and waits until the tape becomes ready before returning control to the calling program.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

EXIT MODE - RTS when TAPWRK = 0

SUBROUTINES CALLED - SSP16, WAIT

DYNAMIC VARIABLES USED - WLTOP+2, TAPBSY

VARIABLES TO BE PRE-INITIALIZED - WLTOP+2, TAPBSY

DETAILED DESCRIPTION -

This routine first removes any pending display program from the wait list so that its upcoming message will not be obliterated before it can be read. Then the message "**WORKING**" is displayed. The routine then waits for about 2 s and checks the TAPBSY flag. If the flag is still non-zero the routine waits again. When the tape is finally ready control returns to the caller by falling through the BNE TAPW1 instruction.

PROGRAM NAME - MSTLCL

BRIEF DESCRIPTION - Outputs the master/local selection message and inputs the response. Control is then transferred to MST or LCL.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - GETDTA

DYNAMIC VARIABLES USED - INPFLG

VARIABLES TO BE PREINITIALIZED - None

DETAILED DESCRIPTION -

This routine enables IRQ interrupts for the first time via a CLI instruction. It then outputs the message "MST=1, LCL=2: A=?" and specifies by the LDAA #1 that one keypad response is expected. The operator types in either a "1" EXEC or a "2" EXEC to determine a master or local start, respectively. No check is made to make sure a "2" was input. If it was not a "1," local is selected. The flag INPFLG indicates the type of initialization for future systems which actually communicate with a master station.

PROGRAM NAME - MST

BRIEF DESCRIPTION - Initializes input and output data blocks from data stored in ROM.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - SSP16, SSWAIT

DYNAMIC VARIABLES USED - INPPNT, IIDPNT, MSTX, MSTX1, MSTX2, OUTPT, NINPS, NOUTS, KWSCL, KVASCL

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine is entered if the master initialization mode is selected. The routine first displays the message "MASTER SELECTED" for a short time. The first part of the program initializes the input data block headers. IIDPNT points to the header and is incremented after each byte is put in the header. INPPNT points to the table IHEAD which contains the addresses of data areas containing initialization information for all header blocks. Therefore, an LDS 0,X from IHEAD gets the address of the initialization data in X. The end of a header is determined by counting down from IIDHSZ which equals the number of bytes in a header. The end of all headers is determined by the value of the NEXT pointer in the current header. If this value is zero, there are no more headers following. Once all headers are done, the program branches to MST3.

MST3 initializes INPTBL, which contains all the addresses of the individual input-data blocks. INPPNT points into this table and MSTX points into the table IADDR which contains the respective initialization addresses. Transfer from IADDR to INPTBL continues until a high-order zero byte is detected in IADDR. The program then branches to MST5.

MST5 starts initialization of the individual input-data blocks. INPPNT points into INPTBL which contains the addresses of all input-data blocks. MSTX points into the IDTA table which contains the addresses of all initialization data areas for the input-data blocks. MSTX1 contains the address of one input-data initialization-block area for a single pass through this phase of the program.

During this phase, MSTX2 contains the address of the corresponding input-data block being initialized. MST5 is terminated when a zero address is found in the IDTA table and the program branches to MST9.

MST9 initializes the CVAL byte of each input-header block by reading the current value of PIA whose address is in the header at RRADDR. This phase is terminated when the NEXT pointer of the current header is zero.

The next phase of the program initializes the output-data blocks. OUTPT points into OUTTBL which contains the addresses of all output-data blocks. MSTX contains the address of one output-data block during each loop of the program. During this pass MSTX2 contains the address of the corresponding initialization data. MSTX1 points into the table containing the addresses for these initialization data areas. This phase is terminated by a zero address in the ODTA table and the program branches to MST20 and initializes the number of inputs, outputs, and KW and KVAR scale factors. The program then terminates by jumping to IOD9.

PROGRAM NAME - LCL

BRIEF DESCRIPTION - Displays "LOCAL SELECTED" message and requests operator input to initialize the number of meter inputs and outputs and the KW and KVAR scale factors.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - SSP16, SSWAIT, MNUMBS, USE3

DYNAMIC VARIABLES USED - NINPS, NOUTS, KWSCL, KVASCL

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine displays the message "LOCAL SELECTED" and then proceeds to interactively query the operator for certain initialization information.

The program requests the number of meter inputs and outputs and stores the results in binary in NINPS and NOUTS, respectively. Next the KW and KVAR scale factors are input and stored in BCD in 2-byte words at locations KWSCL and KVASCL, respectively.

PROGRAM NAME - IID

BRIEF DESCRIPTION - Queries the operator for information necessary to initialize the meter input-data blocks.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - BCDASC, POW1, GETDTA, POWERT, SCALEF, INPDTA, BCDBIN, ASCBCD, SSCLR, SSP, INP, TRY

DYNAMIC VARIABLES USED - NINPS, IIDCNT, IIDPNT, PIAPNT, IIDRR, INUMB, INPPNT, INPTOP, NOUTS, IIDX1, IIDA

VARIABLES TO BE PRE-INITIALIZED - NINPS, NOUTS

DETAILED DESCRIPTION -

This routine initializes all input-data blocks by querying the operator for information. The first part of the routine initializes a set of pointers as follows:

IIDCNT = Number of inputs remaining to be initialized

IIDPNT = Pointer into specific input header block

PIAPNT = Pointer into table containing input PIA addresses

INPPNT = Pointer into table containing pointers to individual input-data blocks

INPTOP = Address of input-data block currently being initialized

IIDRR = Number of inputs left to initialize for a specific header

INUMB = ASCII value of current input being initialized

These values are updated on successive passes through the program.

The output-message block BUF5 is first initialized by filling it with the ASCII values "I#" followed by the storage of a space at BUF5+4. The value of the actual input is stored in the buffer in the IID2 portion of the program.

The header portion of the block is next initialized with the PIA address and current value of PIA. The pointers to individual data blocks and the NEXT pointer are initialized as they are computed in the rest of the program.

The first input-data block is placed directly after the fixed header. This value is saved both in the header by IIDPNT and in INPTBL by INPPNT.

Now data are ready to be acquired and put in the input-data block whose address is in INPTOP. The routine POW1 displays the message "I#01 POWER FLOW" for about 2 s and then GETDTA requests the direction by the message "IN=1, OUT=2: A=?" The value input is converted from ASCII to BCD and saved in the input-data block. A similar scenario is followed for the power type, input scale factor, and maximum pulse time. These values are all input and stored in the respective input-data block.

The attached output information is a variable-length list and may require several loops to process all the data. The message "I#01 OUTPUTS=??" is displayed and the operator may input a string of two-digit numbers separated by commas. A zero-zero input at any time will cause any subsequent outputs to be ignored. All values are input by the initial call to INP. On return from INP the X register contains the address where the input data can be found. The groups of two-digit ASCII numbers are first converted to BCD by ASCBCD and then converted to binary by BCDBIN and temporarily saved at location IIDA. If the number specified is not within the range of outputs expected, a "PLEASE TRY AGAIN" message will be displayed and a new request for outputs will be made. It is important to note that any outputs processed up to this time need not be repeated since they have already been saved in the input-data block. This process continues until a zero buffer terminator is found or a zero-zero output reference is found. In either case the program branches to location IID5.

IID5 starts checking to determine if all inputs have been processed. If so, the program branches to IOD. Otherwise, INUMB is BCD incremented to update the input number for display. INPTOP is then updated to point to the first location following the input-data block just completed. This value will either be the address of the next input-data block or a new header depending on the value of IIDRR. INPPNT and IIDPNT are also updated to point to the next address location in their respective blocks. If a new header is to be initialized the program branches to IID8 and then jumps back to IID1. If it is time to do another input, then a jump to IID2 is executed.

PROGRAM NAME - IOD

BRIEF DESCRIPTION - Queries the operator for information necessary to initialize the meter output-data blocks and certain system parameters.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - BCDASC, POWERT, SCALEF, GETDTA, ASCBCD, USE3, SSP16, HEADER, SSWAIT

DYNAMIC VARIABLES USED - NOUTS, IODCNT, PIAPNT, OUTPT, ONUMB, IODRR, OUTTOP, OUTMAS, SOFTST, DEMPER, CLKDAY, CLKHRS, CLKMIN, CLKSEC, ID

VARIABLES TO BE PRE-INITIALIZED - NOUTS

DETAILED DESCRIPTION -

This routine initializes all output-data blocks and selected system parameters by querying the operator for information. The first part of the routine initializes a set of pointers as follows:

IODCNT =	Number of outputs remaining to be initialized
PIAPNT =	Pointer into table containing output PIA addresses
OUTPT =	Pointer into table containing pointers to individual output-data blocks
IODRR =	Number of outputs left to initialize for a given PIA
OUTTOP =	Address of output-data block currently being initialized

OUTMAS = A bit mask with a 1 in the position of the current PIA output bit

ONUMB = ASCII value of current output being initialized

These values are updated on successive passes through the program.

The output message buffer is the same as for inputs so only the ASCII letter "O" need be added to the buffer.

Next the proper PIAOT table entry is computed according to the number of the output being initialized. Since one PIA can accommodate eight outputs, the pointer into the PIAOT table is incremented by two for each increment of eight that divides the current output number. The computed pointer is then used to access the PIA address which is stored in the output-data block.

The power type, output scale factor, and OUTMSK are then acquired or computed and stored in the output-data block whose address is in OUTTOP.

If all outputs have been done, the program branches to IOD9. Otherwise, OUTTOP is updated to point to the next output-data block. Then ONUMB is BCD incremented. If all outputs are done for the current PIA, control transfers to IOD1. Otherwise, control transfers to IOD4.

Once all outputs are done the program reaches IOD9 where the soft-restart flag is initiated. If the reset switch is pushed after this point, control will transfer to IOD99 after some preliminary initialization instructions.

IOD99 starts the initialization of certain system parameters. The demand period is initialized first. GETDTA displays the message "DEMAND PERIOD=??" and inputs the two-digit ASCII response. The response is converted to packed BCD and stored at DEMPER. The Julian date, time, and station ID are initialized in a similar manner; the message "**WORKING**" is then displayed and a header record is written onto the tape.

PROGRAM NAME - STR

BRIEF DESCRIPTION - Initializes the start mode of the system to be synchronized, immediate or triggered.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - SSP16, SSWAIT, GETDTA, TRY, DEMGO, AWAITS, NEWFRZ, SYSUP

DYNAMIC VARIABLES USED - FRZSEC, FRZMIN, FRZHRS, INIT, CLKSEC, CLKMIN, CLKHRS, INIT1

VARIABLES TO BE PRE-INITIALIZED - INIT, INIT1

DETAILED DESCRIPTION -

This routine first displays the message "SELECT START" for 2 s. The message "S=1, G=2, T=3: A=?" follows. A "1", "2", or "3" input represents a synchronized, go-immediate, or triggered start, respectively.

If a "1" is input, control transfers to STR1. The routine DEMGO inputs the hours, minutes, and seconds when the system is to start. These values in BCD are stored at FRZHRS, FRZMIN, and FRZSEC. The message "AWAITING SYNC" is then displayed by the AWAITS routine and the software clock is allowed to increment for the first time on the next clock interrupt, because the INIT flag has been set non-zero. The program then loops by monitoring INIT1. When the synchronized time is reached (when CLKHRS, CLKMIN, and CLKSEC equal their counterparts FRZHRS, FRZMIN, and FRZSEC), a freeze cycle will be initiated. The end of this cycle sets INIT1 non-zero and the program will escape from the loop and display the message "SYSTEM UP". The program will then transfer control to RINO to start monitoring the Form-C relay inputs.

If a "2" was selected as the start mode the program will branch to STR4. FRZHRS is initialized since it does not get changed by NEWFRZ in some circumstances. The routine NEWFRZ will then initialize the conditions for the next freeze period which will happen in a DEMPER time from the current time. The flags INIT and INIT1 are then set non-zero to allow the software clock to increment and to allow input Form-C relays to be monitored. The program then displays the message "SYSTEM UP" and transfers control to RINO.

If a "3" was selected as the start mode the program will branch to STR2. DEMGO will input the hours, minutes, and seconds of the time that the system is to start when an input trigger occurs. These BCD values are stored in both FRZ and CLK time locations. The system then displays the message "AWAITING SYNC" and starts to monitor the input trigger control bit. When this bit goes to "1" the routine branches to STR5 and executes a sequence similar to the go-start procedure.

PROGRAM NAME - RIN

BRIEF DESCRIPTION - Checks the state of input Form-C relays. If there is a complete Form-C state change, pulse data are incremented and ROT is processed.

ON ENTRY - NA

ON EXIT -

A REGISTER - NA

B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - Same as on entry

EXIT MODE - Always exits by jumping back to beginning of itself or by branching to ROT which returns only to RIN.

SUBROUTINES CALLED - FIFO, OUTADR

DYNAMIC VARIABLES USED - INIT1, RINPNT, RINX1, RINA, RINB, RINX, CLKMIN, ROTX

VARIABLES TO BE PRE-INITIALIZED - RINFWA, RINLAST, INIT1

DETAILED DESCRIPTION -

This routine first executes the FIFO routine to see if any routines are awaiting execution. After a pending routine is executed, or if no routines are pending, control is returned to RINO. Here the INIT1 flag is checked. If it is non-zero, the routine continues to monitor inputs. Otherwise, the routine loops until INIT1 is non-zero.

On each entry into RIN1, the X register contains the address of the header of an input-data block. If this address is zero, the program restarts at RIN. Otherwise, RINX1 is initialized to point into RINTBL. Then the PIA address is fetched from the current input header and the current value of this PIA is compared with the CVAL value in the header. If they are not equal, there is a possibility that a Form-C pulse has occurred and control transfers to RIN5. Otherwise, the next header pointer is fetched from the current header block and control resumes at RIN or RIN1.

If there is a possible pulse the compared value is saved with a "1" in every bit where there is a difference between the current value of the PIA and the CVAL value. The B register is initialized to count the number of relays tested per PIA. If the two low-order bits are both 1's, then there is a definite Form-C pulse and control transfers to RIN7. Otherwise, the previous compared value is shifted right two places and the pointer into table RINTBL is incremented accordingly. Control returns to RIN6 to check again for a positive Form-C pulse, until four relays have been checked or until no further possible pulse conditions exist.

RIN7 is executed if a definite Form-C pulse is found. The B register contains a number from 0 to 3 which indicates which position within the PIA caused the pulse. The current value of the PIA, as found in the CVAL location of the input-data block, is updated to reflect the Form-C state change. That is, the two Form-C related bits are complemented and stored at the CVAL location. Then the address of the pointer to the input-data block is computed and stored at RINX. If this address is zero then control transfers to RIN75. Otherwise, the current pulse count is incremented. This is followed by saving the current value of the minutes location, of the software clock, at the LPULT location of the current input-data block. This LPULT update takes place unless the current value of LPULT=FE, in which case the associated meter has been placed out of service or a load drop has occurred.

At RIN711 the input-data block address is stored at RINX and then the OUTPNT list of the input is examined to check if any outputs are attached to the input. If there is at least one output attached, control transfers to RIN72. Otherwise, the relay counter is restored and control transfers to RIN61 where the rest of the relays are checked for a given PIA.

At RIN72 the routine OUTADR is called to compute the address of the referenced output-data block. The computed address is returned in the X register and promptly stored at ROTX. Next, the current fractional pulse counter (SOUTC) is examined and the value of the input pulse is added to this count. This is done by adding the input scale factor (INSCL), since the scale factor is the KWH value of the Form-C pulse that just occurred. The updated fractional pulse count is compared with the output scale (OUTSCL). If SOUTC $\geq$ OUTSCL, it is time for an output pulse and control transfers to ROT. Otherwise, the address at RINX is updated so it will be the proper value to allow examination of the next output reference, if any, in the current input-data block. This loop will continue until all output references have been handled.

PROGRAM NAME - ROT

BRIEF DESCRIPTION - Increments output pulse counter pointed to by input-data tables. Also causes affected PIA bit to be changed.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA (Data passed in ROTX and ROTX1)

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Conditional branch from RIN

ON EXIT -

A REGISTER - SOUTC (High) B REGISTER - SOUTC+1 (Low)

X REGISTER - FWA of output-data area

STACK - NA

INTERRUPT STATUS - Same as on entry

EXIT MODE - BRA RIN73

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - ROTX

VARIABLES TO BE PREINITIALIZED - None

DETAILED DESCRIPTION -

This routine is used to output Form-C pulses and to update the values of SOUTC and NOUTC accordingly. The new value of SOUTC equals the old value of SOUTC minus OUTSCL. NOUTC is BCD incremented by one. The PIA bit associated with the output is then complemented which causes a Form-C state change to occur in the connected relay. If no PIA is attached, the above PIA update does not occur. Control then returns to RIN73 to determine if the given input should cause more than one output pulse.

PROGRAM NAME - OUTADR

BRIEF DESCRIPTION - Returns the address of the output-data block whose number is in the A register on entry.

ON ENTRY -

A REGISTER - Output number B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - Address of output-data block

STACK - NA

INTERRUPTED STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - ROTX

VARIABLES TO BE PREINITIALIZED - None

DETAILED DESCRIPTION -

This routine takes the output number and computes the proper offset into the table OUTTBL, which contains a list of all output-data block addresses. The pointer into this table is stored in IIDX which is followed by an LDX 0,X instruction to get the output-data block address in the X register for the calling program.

9

PROGRAM NAME - IRQ

BRIEF DESCRIPTION - First handler for all IRQ interrupts. Determines the interrupting device and establishes its control linkage.

ON ENTRY -

A REGISTER - NA　　　　B REGISTER - NA

X REGISTER - NA

STACK - Contains interrupt status of program running when IRQ went active low.

INTERRUPT STATUS - Interrupt mask bit set

ENTERED VIA - Interrupt vector from IRQ or SWI interrupts.

ON EXIT -

A REGISTER - Interrupt status　　　　B REGISTER - NA

X REGISTER - Address of driver to service interrupt

STACK - Same as on entry (IRQX1 = Address of PIA or ACIA), (IRQA = Status)

INTERRUPT STATUS - Interrupt mask bit set

EXIT MODE - JMP to driver to handle interrupt

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - IRQX, IRQFLG, IRQX1, IRQX2, IRQA, IRQNUL

VARIABLES TO BE PRE-INITIALIZED - IRQX, IRQFLG, IRQNUL

DETAILED DESCRIPTION -

This is the routine which is entered in case of an IRQ of software interrupt. Its address is stored in the interrupt pointer table at FFF8 and FFF9. If a device interrupt occurs, and the interrupt mask bit Im is clear, this program will be executed.

The program first checks for a tape interrupt since timing constraints place critical-response-time criteria on interrupt handling. If the "IRQA" flag bit in the tape PIA is set, then a tape interrupt has occurred and the value of the A register will be less than zero and the tape handling routine TP1IRQ will be invoked. Otherwise, control will proceed to IRQ0.

At IRQ0 the X register is loaded with the current pointer into the IRQPNT table. This table contains a list of routine addresses followed by a corresponding list of PIA or ACIA control-register addresses which is terminated with a double byte zero. If the pointer to IRQX is at the end of the list, control transfers to IRQ5. Otherwise, the IRQFLG is incremented to keep track of the number of interrupts checked during the current execution of IRQ. If the high-order byte of the routine address pointed to by IRQX is zero, then that routine is busy and control transfers to IRQ3. Otherwise, the control register of the associated PIA or ACIA is examined. If the most significant bit is zero then an interrupt has not occurred and control passes to IRQ3. Otherwise, the address of the PIA or ACIA and the current status of the interrupting device are saved for the device-interrupt handler at locations IRQX1 and IRQA, respectively. The IRQFLG is cleared for subsequent executions of IRQ and the pointer into the IRQPNT table is updated to point to the next entry. Therefore, the IRQPNT table is always examined in a circular fashion so that no device, except for the tape drive, gets priority treatment. After all values are updated, control is transferred to the device handler of the interrupting program. If more than one interrupt is pending, the other non-handled interrupts will be handled on subsequent executions of IRQ provided the devices do not reset their own interrupt flag bits.

At IRQ3 the pointer into IRQPNT is updated and the rest of the list is searched.

At IRQ5 it has been determined that the end of the list has been reached. If all interrupts have been checked, then IRQFLG is greater than or equal to #NINT-1 and this indicates that a false interrupt occurred. The false-interrupt counter is incremented, IRQFLG is set to zero, and an RTI is executed to finish the interrupt processing. If all possible interrupting devices have not been checked, then processing continues at IRQ1.

PROGRAM NAME - TTYIRQ

BRIEF DESCRIPTION - Handles interrupts for Teletype input or output data.

ON ENTRY -

A REGISTER - Status of ACIA B REGISTER - NA

X REGISTER - TTYIRQ

STACK - Contains status of interrupted program.

INTERRUPT STATUS - Interrupt mask bit set.

ENTERED VIA - IRQ program

ON EXIT -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA on normal exit; address of recall program for recall exit.

STACK - Last interrupt status cleared on normal exit; stack not changed on recall status.

INTERRUPT STATUS - Set to status of interrupted program.

EXIT MODE - Normal via RTI; recall via JMP XQT

SUBROUTINES CALLED - WAIT, XQT

DYNAMIC VARIABLES USED - IRQPNT+6, IRQX1, TINLST, TINPNT, TINRCL, TOTPNT, TOTRCL

VARIABLES TO BE PRE-INITIALIZED - IRQPNT+6, TINLST, TINPNT, TINRCL, TOTPNT, TOTRCL

DETAILED DESCRIPTION -

Since this routine handles both read and write interrupts from the Teletype, the first part of the routine determines the interrupting device. The previously read ACIA status is in the A and B registers. The device is set busy by clearing the high-order address byte in the IRQPNT table. Next the ACIA status is checked for errors. If any error flags are set, control transfers to TTY8. Otherwise, the two low-order bits of the status are checked to determine if it is time for a write or if a read has happened. If both read and write registers are ready, the read will take precedence. The routine branches to TTYIN or TTYOUT if the interrupt was read or write, respectively.

TTY8 is reached only in case of an error. In this case the device-interrupt flag bit is cleared by a device read and then control transfers to ENDTTY.

Input from the Teletype is handled at TTYIN. The character is input from the ACIA receive-data register and immediately echoes the character back to the ACIA transmit data register to cause the character to be printed on the Teletype. The input character is then stored in the input buffer. If this buffer is full, control transfers to TTY6. Otherwise, the character is checked to determine if it is a carriage return. If so, control transfers to TTY6. Otherwise, the input buffer pointer is incremented and control transfers to ENDTTY.

TTY6 is reached if a carriage return or end-of-buffer event occurs. An FF character is stored at the end of the buffer and a line feed character is output as soon as the transmit-data register is ready. The Teletype handler is then set ready and control transfers to the Teletype input recall address stored at TINRCL.

If an output interrupt is detected, control transfers to TTYOUT. The next character to output is fetched from the output buffer. If it is a zero byte, then the buffer has been emptied and control transfers to TTY7. Otherwise, the pointer into the buffer is incremented and the character is output to the ACIA transmit-data register. If the character output was a line feed, then the program idles for three character times to allow for Teletype carriage travel. Control then passes to ENDTTY.

PROGRAM NAME - TP1IRQ

BRIEF DESCRIPTION - This routine handles tape read and write interrupts.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Contains status of interrupted program.

INTERRUPT STATUS - Interrupt mask bit set.

ENTERED VIA - IRQ program

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Last interrupt status cleared on exit.

INTERRUPT STATUS - Set to status of interrupted program.

EXIT MODE - RTI

SUBROUTINES CALLED - IRQ0, STOP

DYNAMIC VARIABLES USED - IRQPNT, TAPPNT, TAPFLG

VARIABLES TO BE PRE-INITIALIZED - IRQPNT, TAPPNT, TAPFLG

DETAILED DESCRIPTION -

This routine handles interrupts for tape reads and writes. The interrupt-flag bit is cleared immediately so that the PIA will be ready to receive the next character before the current character is fully processed. Next, the PIA status is checked to see if tape interrupts are enabled. This is done since the flag bit can be set, even though interrupts are not enabled. If the tape interrupts are not enabled, then control returns to IRQ0 to find the real cause of the interrupt.

Next, the handler is set busy and a read or write determination is made. If a write mode is detected, control transfers to TP1I2. Otherwise, a read tape interrupt is processed. The read-pulse-enable bit is set low by the STAB DTATP1 instruction and then a character is read from DTATP2. The read pulse enable is then brought high again after the character has been read. The character is then stored in an input buffer. If the buffer is full, the program branches to ENDTAP. Otherwise, it will update the buffer pointer and then branch to ENDTAP if the buffer terminator is not found. If the buffer is done, its interrupt enable is turned off and any pending interrupt-flag bits are cleared. The tape is then stopped and control transferred to ENDTAP.

If a write interrupt occurs, processing takes place at TP1I2. The next character to be output is fetched from the buffer and output to the tape. The buffer pointer is updated and control transfers to TP1I1 if no further data are to be written. If more data remain, the handler is set not busy at ENDTAP and an RTI is executed.

PROGRAM NAME - TP2IRQ

BRIEF DESCRIPTION - Handles inter-record gap interrupts from tape drive.

ON ENTRY -

A REGISTER - NA　　　　B REGISTER - NA

X REGISTER - NA

STACK - Contains status of interrupted program.

INTERRUPT STATUS - Interrupt mask bit set.

ENTERED VIA - IRQ program

ON EXIT -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Last interrupt status cleared on exit.

INTERRUPT STATUS - Set to status of interrupted program.

EXIT MODE - Control transferred to address in IRGRCL by call to XQT.

SUBROUTINES CALLED - XQT, STOP, IRQ0

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine handles inter-record gap interrupts. The first check in the routine is to determine if inter-record gap interrupts have been allowed. If allowed, control transfers to TP2I1. Otherwise, the interrupt-flag bit is reset and control transfers to IRQ0.

At TP2I1 the tape motion is stopped, interrupts are disabled, and control is transferred to the address specified at IRGRCL by the XQT routine.

PROGRAM NAME - TPBKSP

BRIEF DESCRIPTION - Backspaces tape over one record and stops in inter-record gap.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Contains return address.

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack before return.

INTERRUPT STATUS - NA

EXIT MODE - Jumps to RINO to await interrupt. After interrupt programs, returns to caller by a jump indexed instruction.

SUBROUTINES CALLED - RUNTAP

DYNAMIC VARIABLES USED - TPBX

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine pulls the return address from the stack and saves this address at TPBX. The tape is put into read reverse mode and the run bit is pulsed. The recall address is set to TPB1 and the inter-record gap interrupt is enabled. Control is then transferred to RINO.

When an interrupt occurs execution will be transferred from TP2IRQ to the IRGRCL recall address which has been set equal to TPB1. The previously saved return address is saved at TPBX and control is returned to this address.

PROGRAM NAME - WRTTAP

BRIEF DESCRIPTION - Puts tape control bits in forward and write modes and selects proper data direction in PIA peripheral data register.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of caller

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

The forward and write bits of the tape-status word are set accordingly and the PIATP2 peripheral data register is set up as an output-data register by addressing the associated data direction register and loading it with all 1's. Then the data peripheral is reselected by storing a $2C in PIATP2. Finally any pending interrupt-flag bits are cleared in PIATP1.

PROGRAM NAME - RDTAP

BRIEF DESCRIPTION - Puts tape control bits in forward and read modes.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of caller

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

The forward and read bits of the tape status word are set accordingly and control transfers to the TAPMOD entry point of WRTTAP to set the PIA data direction register for input.

PROGRAM NAME - TAPOUT

BRIEF DESCRIPTION - Outputs a buffer of data to the tape drive and then branches to the background program.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - FIFO stack recall

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

EXIT MODE - Branches to RINO

SUBROUTINES CALLED - ALAR1, WAIT, CHKTAP, WRTTAP, RUNTAP

DYNAMIC VARIABLES USED - WLTOP+2, TAPPNT

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

The initial entry point of this routine is at TAPOUT. The routine first clears any pending interrupt-flag bits in PIATP1 or PIATP2. If the tape is not ready, control will transfer to TAPO which will sound an alarm, wait about 2 s, and try again until the tape is ready.

If the tape is ready, the drive is put in write mode and a pointer to the data in BUF2 is initialized and saved at TAPPNT. TAPFLG is then initialized to indicate that the transfer is not done. Interrupts are then enabled and the tape is started by RUNTAP. The program then enters a wait loop until TAPFLG becomes non-zero. When this happens control will transfer to RINO.

PROGRAM NAME - TAPEIN

BRIEF DESCRIPTION - Inputs a buffer full of data from tape.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack prior to return.

INTERRUPT STATUS - NA

EXIT MODE - Returns to caller by a jump indexed instruction.

SUBROUTINES CALLED - RUNTAP, WAIT

DYNAMIC VARIABLES USED - TAPPNT, TAPFLG, TAPEX

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine saves the calling program return address by pulling it from the stack and saving it at TAPEX. The tape is then started by RUNTAP and the read-bus-enable line is pulsed to initialize the tape drive's read hardware. The input buffer pointer is then initialized, TAPFLG is cleared to indicate that the buffer is not full at BUF2, and tape interrupts are enabled. The routine then enters a wait loop until TAPFLG becomes non-zero. On exit from this loop, control returns to the original caller, whose address is in TAPEX.

PROGRAM NAME - TAPASC

BRIEF DESCRIPTION - Converts BCD data in BUF2 to ASCII data in BUF1 and outputs converted data to TTY.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Contains return address of calling program.

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack prior to exit

INTERRUPT STATUS - NA

EXIT MODE - Returns to caller by jump index instruction unless end of file (EOF) found. In that case jumps to RINO.

SUBROUTINES CALLED - CRLF, HEXAS1, BCDASC, BUF1, BUF2, USE2, TPBKSP

DYNAMIC VARIABLES USED - TAPAX, TAPAX1, TAPAX2, TOTRCL

VARIABLES TO BE PRE-INITIALIZED - BUF2

DETAILED DESCRIPTION -

This routine first initializes a pointer into BUF2 (the buffer to be converted from BCD to ASCII). The first entry in this buffer is examined and the buffer pointer is incremented. If the character has a $FF value, it indicates that a tape EOF record has been detected and control transfers to TAPA9. Otherwise, the X register is initialized to point to the first location in the output buffer to be used to store the ASCII output characters and control continues at TAPA0. Carriage return and line-feed characters are first stored in BUF1 followed by the hexadecimal (hex) value of the record type. The current output-buffer pointer which was updated by CRLF and HEXAS1 is saved at TAPAX1. The input pointer is then loaded into TAPAX and the next BCD characters are fetched. If a $FF character is found, it denotes end of record and control transfers to TAPA8. Otherwise, the two packed BCD characters are converted to ASCII by BCDASC. The converted values are returned in the A and B registers and if they are numerically less than $3A, they are valid BCD numbers and control transfers to TAPA3 or TAPA4 depending on the register being examined. If an illegal BCD character is found, an ASCII blank is inserted in its place. This routine will continue to loop and convert characters until the output buffer is full or a $FF byte is detected.

If the output buffer is full, USE2 is executed to output the full buffer and a recall address is set equal to TAPA91.

If the last buffer is to be output, control transfers to TAPA8. The recall address is set to TAPA92 and the buffer is output by transferring control to TAPA71.

If an EOF condition is found, control transfers to TAPA9. The TPBKSP routine backspaces over the EOF record and control transfers to RINO.

If more data follow, the output routine will return to TAPA91. The output-buffer pointer is initialized to BUF1 and control transfers to TAPA1.

If the last data have been output, control returns to TAPA92 and the routine returns to the original caller of TAPASC.

PROGRAM NAME - TAPCLO

BRIEF DESCRIPTION - Closes a tape file by writing a date-time stamp onto tape.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Contains return address that will be pulled from stack by XQTS return function.

INTERRUPT STATUS - NA

EXIT MODE - Jump to XQTS which will return to caller of TAPCLO.

SUBROUTINES CALLED - XQTS, IDBUF

DYNAMIC VARIABLES USED - BUF2

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

A $FF byte is saved in BUF2 as a record type. IDBUF then fills part of BUF2 with the proper time and date stamp. The BUF2 data are then prepared for output by TAPOUT and the TAPOUT routine is put on the FIFO stack by XQTS.

PROGRAM NAME - KEYIRQ

BRIEF DESCRIPTION - Handles interrupts from keypad.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Contains status of interrupted program.

INTERRUPT STATUS - Interrupt mask bit set.

ENTERED VIA - IRQ routine

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Last interrupt status cleared on RTI exit or by XQT on ATTN exit

INTERRUPT STATUS - Set to status of interrupted program.

EXIT MODE - RTI or JMP XQT

SUBROUTINES CALLED - XQT

DYNAMIC VARIABLES USED - IRQPNT+4, KEYCNT, KEYPNT, BUF4

VARIABLES TO BE PRE-INITIALIZED - IRQPNT+4, KEYCNT, KEYPNT, BUF4

DETAILED DESCRIPTION -

This routine is entered from IRQ if a keyboard interrupt is detected. The handler is set busy and the A- and B-side PIA data are read. Normally these data have all bits set equal to 1 except the bit corresponding to the depressed key. Therefore, if the CMPA #$FF yields 0, this indicates that no keys were depressed on the A side and, therefore, a B-side key must have caused the interrupt.

When the program reaches KEY3 the X register will contain the address of the proper ASCII look-up table for the depressed key. At this point KEYCNT is examined to determine if the interrupt was unexpected. If so, KEYCNT will equal 0, and control will transfer to KEY8. Otherwise, the data input will be shifted right until a 0 bit is detected. Each shift is accompanied by an INX instruction so that when a 0 bit is detected and control transfers to KEY5, the X register will point to the place in the ASCII look-up table. If no bits are 0, then the key that caused the interrupt was released before the data could be input and control transfers to KEY6. Note that if more than one key is depressed, the key corresponding to the bit position closest to the least significant end of the byte will be recognized.

At KEY5 an ASCII value for the key is placed in the buffer pointed to by KEYPNT. If this buffer is full, control transfers to KEY7. Otherwise, the buffer pointer (KEYPNT) is incremented, the key count (KEYCNT) is decremented, the handler is set ready, and control returns to the interrupted program.

If the buffer is full, an error condition exists; control continues at KEY7, the buffer pointer is reset to the beginning of the buffer, and control transferred to KEY6.

If a KEYCNT equals zero, control continues at KEY8. If the system is not currently enabled to monitor meter data (INIT1=0), then the key interrupt is ignored. If INIT1 is non-zero, control transfers to KEY9, the routine is set ready, and the MSGATN routine is placed on the FIFO stack by XQT.

PROGRAM NAME - MSGATN

BRIEF DESCRIPTION - Displays the "TYPE OP-CODE ??" message and inputs response. Control is then transferred to message processor.

ON ENTRY -

A REGISTER - NA

B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - FIFO stack recall

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

EXIT MODE - JMP MSGKEY

SUBROUTINES CALLED - GETDTA

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine is evoked by KEYIRQ, which places it on the FIFO stack. It outputs the message "TYPE OP-CODE ??" to request that the operator type the two-digit op-code desired. Once the inputs have been made, control transfers to MSGKEY for processing of the op-code.

PROGRAM NAME - ADC

BRIEF DESCRIPTION - Enables ADC interrupts.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Wait list recall

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

EXIT MODE - JMP RINO

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine is called off the waiting list periodically to enable ADC interrupts. Once the interrupt is enabled, control transfers to RINO.

PROGRAM NAME - ADCIRQ

BRIEF DESCRIPTION - This routine handles interrupts received from the ADC.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Contains status of interrupted program

INTERRUPT STATUS - Interrupt mask bit set.

ENTERED VIA - IRQ routine

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Last interrupt status cleared on exit.

INTERRUPT STATUS - Set to status of interrupted program.

EXIT MODE - RTI

SUBROUTINES CALLED - IRQ0

DYNAMIC VARIABLES USED - WLADC+2

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine is entered from IRQ if an ADC interrupt is detected. If interrupts are allowed, control transfers to ADC0. Otherwise, control returns to IRQ to find the real cause of the interrupt.

At ADC0 the status bit of the ADC is checked to make sure that a valid data word is still present at the output of the ADC. If so, control transfers to ADC1. Otherwise, an interrupt return is executed.

At ADC1 the data type is checked for KW or KVAR. If KW, then the X register will equal KWDTA, and if KVAR, it will equal KVADTA. The data are then stored at the location pointed to by the X register, and ADC interrupts are disabled. The KW/KVAR select bit is complemented so the opposite input will be checked on the next ADC cycle. The ADC program is then put back on the wait list for 6 s.

PROGRAM NAME - SSCLR

BRIEF DESCRIPTION - Clears the self-scan panel display to all blanks.

ON ENTRY -

A REGISTER - NA	B REGISTER - NA
X REGISTER - NA	
STACK - NA	
INTERRUPT STATUS - NA	
ENTERED VIA - Subroutine call	

ON EXIT -

A REGISTER - NA	B REGISTER - NA
X REGISTER - Unchanged	
STACK - NA	

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - SSBSY

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

Sixteen spaces are output to the display. After each space the routine SSBSY is called to wait until the display is ready for the next character. The character count is maintained in the A register since the A register is not changed by SSBSY.

PROGRAM NAME - SSBKSP

BRIEF DESCRIPTION - Backspaces self-scan display n spaces.

ON ENTRY -

A REGISTER - Number of backspaces B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - Unchanged

STACK - NA

INTERRUPT STATUS - Unchanged

EXIT MODE - RTS

SUBROUTINES CALLED - SSBSY

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

The data-present pulse is disabled so that no data-present pulses will be generated by the STAB DTASS. This is because the backspace pulse acts as its own data-present signal. However, the data-taken signal must be received before the next backspace or other character can be sent. Therefore, the data-taken pulse is awaited by SSBKSP. When the display is ready again, the data-present pulse is enabled and control returns to the caller.

PROGRAM NAME - SSUBLK

BRIEF DESCRIPTION - Unblanks self-scan display.

ON ENTRY -

A REGISTER - NA　　　　B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - Unchanged　　　　B REGISTER - NA

X REGISTER - Unchanged

STACK - Unchanged

INTERRUPT STATUS - Unchanged

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

The data-present pulse is disabled as in SSBKSP and a blanking control byte is sent to the display. After the blanking signal, the data-present pulse is enabled.

PROGRAM NAME - SSP and SSP16

BRIEF DESCRIPTION - Outputs character data to self-scan display.

ON ENTRY -

A REGISTER - Character counter B REGISTER - NA

X REGISTER - Address of output buffer

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Unchanged

INTERRUPT STATUS - Unchanged

EXIT MODE - RTS

SUBROUTINES CALLED - SSBSY, SSUBLK

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

SSP16 sets the A register to 16 and then proceeds to SSP, which is entered with the A register containing the number of characters to be displayed. SSP is also a callable entry point used to output an arbitrary number of characters to the display.

On entry to SSP or SSP16 the X register is set to the address of the buffer which contains the ASCII character string to be displayed. Characters are fetched from this buffer and the two most significant bits are set to 1's to blank the display and to disable the backspace function. The low-order six bits select one of 64 displayable character codes.

If the A register was zero on entry, control transfers to SSP6 where the next output character is checked. If it is zero the routine is done and control transfers to SSP8. Otherwise, the routine loops until a zero character is found.

If the A register represents a legitimate character count, then this count is decremented and looping continues until a zero count is found.

After each character is output to the display the data-taken pulse is awaited by SSBSY. After the last character the display is unblanked by SSUBLK.

PROGRAM NAME - SSBSY

BRIEF DESCRIPTION - Waits until self-scan display is done with current character and then returns to caller.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - Unchanged   B REGISTER - Unchanged

X REGISTER - NA

STACK - Push and pull done on stack; therefore, stack unchanged.

INTERRUPT STATUS - NA (checks pseudo-interrupt flag only)

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine waits for the data-taken flag to be raised in PIASS. The B register is saved for the stack so it can be restored on exit from the routine. The routine loops until the interrupt-flag bit is raised indicating that the data-taken pulse has occurred. The B register is then restored and control is returned to the calling program.

PROGRAM NAME - FIFO

BRIEF DESCRIPTION - Pops first address from FIFO stack and jumps to that address.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Regular stack unchanged; FIFO stack has first entry removed if there was one.

INTERRUPT STATUS - Unchanged

EXIT MODE - If no entries on stack, then RTS; otherwise, JMP to address first on stack.

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - XQTCNT, WAITSP (stack), FIFX

VARIABLES TO BE PRE-INITIALIZED - XQTCNT

DETAILED DESCRIPTION -

This routine checks the number of programs currently on the FIFO stack by examining XQTCNT. If this value is zero, no programs are on the stack and control transfers to FIF3. Otherwise, the first address on the stack is fetched from WAITSP and saved temporarily at FIFX. The address of the stack is then loaded into the X register and the number of programs on the stack is decremented at FIF1. If the count is now zero, then there are no more programs on the stack and control transfers to FIF2. Otherwise, all addresses on the stack are brought one location closer to the top of the stack. Therefore, the program that was second on the stack on entry would be first and the Nth program would be N-1.

FIF2 is reached when the stack has been updated accordingly. The address of the program that was on the top of the stack on entry is retrieved from FIFX and the stack counter is reduced by one. Control then transfers to the program removed from the stack.

PROGRAM NAME - XQT and XQTS

BRIEF DESCRIPTION - Places program whose address is in X on the FIFO stack to await execution.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - Address of program to be added to FIFO stack

STACK - Return address of calling program if XQTS entry

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call to XQT or XQTS

ON EXIT -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Regular stack pulls return address from stack on exit. FIFO stack has address appended to its end.

INTERRUPT STATUS - NA

EXIT MODE - RTI if XQT entry; RTS if XQTS entry.

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - XQTFLG, XQTX, XQTCNT

VARIABLES TO BE PRE-INITIALIZED - XQTFLG, XQTCNT

DETAILED DESCRIPTION -

This routine has two entry points. It is entered at XQT if it is called by an interrupt-handling routine that requires an RTI exit. It is entered at XQTS if it is called as a standard subroutine. The flag XQTFLG is set non-zero for an XQTS entry and has been preset to zero for an XQT entry.

In either type of entry, the X register is set to the address of the program which is to be added to the FIFO stack and the X register is temporarily saved at XQTX. The address of the stack is placed in the X register and incremented by two for each program currently on the stack, as determined by the XQTCNT counter. When the X register points to the first empty location at the bottom of the stack, control is transferred to XQT2. The address to be placed on the stack is fetched from XQTX and placed on the stack. The program counter, XQTCNT, is incremented by one and the entry flag is placed in the A register.

If the entry was at XQT, control transfers to XQT3. If it was at XQTS, the entry flag is cleared and an RTS instruction is executed.

PROGRAM NAME - DEMGO

BRIEF DESCRIPTION - Outputs the demand-go message and then inputs the time and pushes BCD hours, minutes, and seconds onto the stack.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - BCD seconds B REGISTER - NA

X REGISTER - Return address

STACK - Return address pulled from stack prior to exit and HRS, MINS, and SECS pushed onto stack in that order.

INTERRUPT STATUS - NA

EXIT MODE - Return to caller by a jump indexed instruction

SUBROUTINES CALLED - ASCBCD, GETDTA

DYNAMIC VARIABLES USED - DEMX

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

The return address is pulled from the stack and saved at DEMX. The message "DEMAND GO=??????" is then output to the display and a six-digit ASCII response is returned by GETDTA. On return from GETDTA the X register points to the buffer containing the ASCII digits. The ASCII data are converted to packed BCD by ASCBCD which returns the packed BCD result in the A register. Therefore, on exit the stack contains 3 bytes of data: (1) the BCD seconds, (2) the BCD minutes, and (3) the BCD hours, corresponding to the top, the second, and the third entry in the stack.

PROGRAM NAME - MULT, MULTC, MULTL, and MULTLC

BRIEF DESCRIPTION - Multiplies two unsigned binary single or double precision numbers and either adds or does not add a constant value to result.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call (XA = multiplier, YA = multiplicand, Q = product)

ON EXIT -

A REGISTER - NA

B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - PREC, XA, YA, TA, Q

VARIABLES TO BE PRE-INITIALIZED - TA, XA, YA, Q (if MULT or MULTL entry)

DETAILED DESCRIPTION -

This routine has four entry points to facilitate binary single or double-precision multiplication of unsigned binary numbers with and without an additional constant addition. The functions performed by each routine are as follows:

MULT--double-precision multiply with constant added

$$Q \text{ (4 bytes)} + \left[XA \text{ (2 bytes)} \times YA \text{ (2 bytes)}\right] = Q \text{ (4 bytes)}$$

MULTC--double-precision multiply

$$XA \text{ (2 bytes)} \times YA \text{ (2 bytes)} = Q \text{ (4 bytes)}$$

MULTL--single-precision multiply with constant added

$$Q_{2,3} \text{ (2 bytes)} + \left[XA_1 \text{ (1 byte)} \times YA_1 \text{ (1 byte)}\right] = Q_{2,3} \text{ (2 bytes)}$$

MULTLC--single-precision multiply

$$XA_1 \text{ (1 byte)} \times YA_1 \text{ (1 byte)} = Q_{2,3} \text{ (2 bytes)}$$

At the product-cleared entry of MULTC and MULTLC the product Q is cleared to zero. At every single-precision entry point, the precision flag, PREC, is set to \$80 to equal single precision.

With the appropriate pre-conditions set, all entry points proceed to MULT to perform the actual multiplication desired. If the multiplicand or multiplier is equal to zero, the product also is zero and control branches to MUL5. Otherwise, the well-known add/shift method is used to perform the binary multiplication of unsigned binary numbers. A bit counter of 16 is loaded into the X register and the least significant byte of the multiplier is loaded into the A register. If the A register is zero, control transfers to MUL6 to perform a fast eight-bit shift.

At MUL1 the multiplier is successively shifted right and the low-order bit is examined. If the bit is a 1, then an add operation is performed. If this bit is not set, a shift operation is performed.

The point SHIFT is reached whenever the low-order bit of the multiplier is a zero. The high- and low-order bytes of YA (the multiplicand) and TA (an extension of the multiplicand used to maintain a double-length shift register) are shifted left one place. For this operation the TA and YA locations can be viewed as a 32-bit shift register of the form [TA,TA+1,YA,YA+1].

When the 32 bits have been shifted one bit to the left, control returns to MUL2 to update the bit counter. If all 16 bits are done, control transfers to MUL4. If eight bits are done, the PREC flag is examined, and if a single-precision multiply is called for, control transfers to MULT4. Otherwise, control transfers to MUL1 and the loop repeats until all 16 bits are done.

The ADD routine is entered to perform the operation [YA (2 bytes) TA (2 bytes)] + Q (4 bytes) = Q (4 bytes). After an add sequence the shift sequence is always performed.

MUL3 is reached when it is time to operate on the most significant byte of the multiplier. If this value is zero, then the routine has finished multiplying, and control transfers to MUL4. Otherwise, the add/shift loop at MUL1 is entered.

MUL4 and MUL5 clear the temporary register TA and PREC, respectively and then return to the caller with the multiplication result in Q.

PROGRAM NAME - BCDMLT

BRIEF DESCRIPTION - Multiplies a 16-bit unsigned binary number by a 6-bit BCD number and returns a 12-digit BCD result.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call (XA = 16-bit binary multiplier; YA = 3-byte BCD multiplicand; and Q = 6-byte BCD result)

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - XA, YA, TA, Q

VARIABLES TO BE PRE-INITIALIZED - XA, YA

DETAILED DESCRIPTION -

This routine multiplies a 6-digit BCD number, stored at YA, by a 16-bit binary number, stored at XA, and returns a 12-digit BCD result starting at location Q.

The routine first clears the product storage locations at Q and the temporary register TA. A 16 is loaded into the X register to act as a bit counter and the least significant byte of the binary multiplier is loaded into the A register.

At BMLT1 a loop begins which successively examines the least significant bit of the multiplier. If this bit is 1, a shift and add occurs, and if it is a 0, only a shift occurs. The sequence through BMLT3 is equivalent to the MULT routine from MUL1 through MULT6 except for the fact that not as many 0 checks are made.

The routines SHFTB and ADDB are also similar to their MULT counterparts SHIFT and ADD. The difference is that they perform BCD shifts by a doubling and BCD adjusting operation and perform adds in BCD mode.

PROGRAM NAME - BINBCD

BRIEF DESCRIPTION - Converts a 16-bit binary number to a five-digit BCD number.

ON ENTRY -

A REGISTER - MS byte of binary number

B REGISTER - LS byte of binary number

X REGISTER - None

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA

B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program pulled from stack before exit; on exit, stack has MS byte of result of SP, next byte at SP+1, and LS byte at SP+2.

INTERRUPT STATUS - NA

EXIT MODE - Returns to calling program by jump-indexed instruction

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - TTD, OTD, HD, TD, RET

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine converts a 16-bit binary number to a five-digit packed BCD number. The method used is to successively subtract smaller powers of 10, starting at 10 , from the number to be converted. The number of subtractions used for a particular power of 10 becomes the coefficients in the series $(C \times 10^4)+(C_2 \times 10^3)+(C_3 \times 10^2)+(C_4 \times 10^1)+(C_5 \times 10^0) =$ result.

The routine first removes the return address from the stack and stores it in RET, so that the stack can be used to return the result of the conversion. The A register containing the most significant byte to be converted is temporarily stored in TTD.

At BINB1 the A register contains the most significant byte of the binary number and the B register, the least significant byte. If the subtraction of 10,000 yields a result less than zero, enough values of 10,000 have been subtracted to determine the coefficient C, which is in the X register. This value is stored at TTD. The number is restored to the value before the last subtraction and 1000 is successively subtracted from this restored value. A similar procedure is repeated and the thousands digit, hundreds digit, and tens digit are stored at OTD, HD, and TD, respectively.

The values are then rotated to the proper byte positions and pushed onto the stack for the use of the calling program. Return is by a jump-indexed instruction with reference to the previously saved return location.

PROGRAM NAME - TRY

BRIEF DESCRIPTION - Outputs the message "TRY AGAIN" to the self-scan panel and waits for 2 s.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack before exit.

INTERRUPT STATUS - NA

EXIT MODE - Returns to caller by jump-indexed instruction.

18

SUBROUTINES CALLED - SSP, SSWAIT

DYNAMIC VARIABLES USED - TRYX

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine pulls the return address from the stack and saves it at TRYX to avoid possible conflicts in subroutine linkage that may be caused by wait-list recalls. The message "PLEASE TRY AGAIN" is displayed for about 2 s and control is then returned to the caller.

PROGRAM NAME - BLANKS

BRIEF DESCRIPTION - Outputs a number of blanks, specified in the A register to the display.

ON ENTRY -

A REGISTER - Number of blanks B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - Unchanged

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - SSP

DYNAMIC VARIABLES USED - BLAX

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

The X register is saved on entry so that the same value of X will be returned to the caller on exit from the subroutine. The A register contains the number of blanks to be output by SSP from location SPACE.

PROGRAM NAME - AWAITS

BRIEF DESCRIPTION - Displays the message "AWAITING SYNC".

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - SSP16

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

Displays the message "AWAITING SYNC" by calling SSP. The RTS instruction in SSP16 will cause control to be returned to the caller of AWAITS.

PROGRAM NAME - SYSUP

BRIEF DESCRIPTION - Outputs the message "SYSTEM UP" and initializes the wait-list times for ADC and CHKPUL routines.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINE CALLED - SSP16

DYNAMIC VARIABLES USED - KEYPNT, WLADC, WLPUL

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine initializes the keypad input-buffer pointer to BUF4 and places the routines ADC and CHKPUL on the wait list for 25 s each. (Their addresses on the wait list were initialized previously.) The message "SYSTEM UP" is then displayed by SSP16, which returns to the caller of SYSUP.

PROGRAM NAME - NEWPER

BRIEF DESCRIPTION - Moves current pulse data to last pulse data storage and updates total pulse count; also, updates the maximum demand values and inputs next freeze time.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - Interrupt mask bit set by NMI.

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack on exit. (Stack was also used for temporary storage but was returned to the same state in which it was found.)

INTERRUPT STATUS - Interrupt mask bit still set by NMI.

EXIT MODE - RTS

SUBROUTINES CALLED - NEWFRZ

DYNAMIC VARIABLES USED - INPTBL, NINPS, INPPNT, OUTTBL, NOUTS, OUTPT, WLSTAT+2

VARIABLES TO BE PRE-INITIALIZED - INPTBL, OUTTBL, NINPS, NOUTS

DETAILED DESCRIPTION -

This routine is called at the end of the demand period to update metering counters and produce reports of the period as required.

The X register is set to point to the table containing pointers to each individual input-data block (INPTBL). The B register is initialized to count the number of inputs to be processed. The pointer into INPTBL is saved in INPPNT so it may be updated to sequence through the table. The input counter is saved on the stack so that it may be easily recalled at the end of each loop to check the input count.

The current demand-period pulse count (NPULC) is transferred to the last demand-period pulse count (NPULL) and then NPULC is cleared so it is ready to accumulate the next demand period's data. The total pulse count for the billing period (NPULT) is then incremented by the value of NPULL, which now contains the number of pulses for the demand period. Since all values are in BCD, a decimal adjust must be performed for each addition.

Next, the current maximum demand value (MAXDEM) is checked against the number of pulses in the just completed demand period. The subtract instructions will not necessarily return the proper difference between MAXDEM and NPULL, since they are in BCD, but if the carry is cleared, then MAXDEM is greater than or equal to NPULL and control transfers to NEW11. Otherwise, the pulse counter for the demand period just ended is the new maximum demand and it is stored at MAXDEM.

At NEW11 the input counter is checked and control transfers to NEW2, if all inputs are done. Otherwise, the pointer into the table INPTBL is updated to point to the next entry, and the program loops through NEW1 until all inputs are done.

At NEW2 a sequence similar to the above input update is performed for all outputs. After the output values are updated, control transfers to NEW4 where the bit controlling the freeze pulse Form-A relay is complemented to close the relay. The routine NEWFRZ is then called to compute the next freeze time. The STATUS routine is then placed on the wait list for 10 ms so that it will be recalled after the freeze sequence terminates. The routine then returns to its caller.

PROGRAM NAME - NEWFRZ

BRIEF DESCRIPTION - This routine updates the freeze-time, clock entries to equal the time of the next freeze.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - Interrupt mask bit set if called from NEWPER.

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - Interrupt mask bit set if called from NEWPER.

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - DEMPER, CLKMIN, FRZMIN, CLKHRS, FRZHRS, CLKDAY

VARIABLES TO BE PRE-INITIALIZED - DEMPER, CLKMIN, CLKHRS, FRZHRS (if CLKMIN + DEMPER $<$ 60), CLKDAY

DETAILED DESCRIPTION -

The current length of the demand period in BCD minutes is added to the current number of BCD minutes of the software clock. If the carry is set by this operation, then CLKMIN + DEMPER is greater than 99 min and control transfers to NEW5. Otherwise, the value is transferred to the B register for safekeeping and a BCD 40 is added to the previous result. If this previous result was from 60 to 99 min, inclusive, then the addition of 40 will set the carry bit and control will transfer to NEW6. If the carry was not set, then the original result of CLKMIN + DEMPER was from 0 to 59 min, and it only remains to save this value at FRZMIN and exit at NEW99.

At NEW5 a time greater than 99 min is handled. On entry to this point the A register equals CLKMIN + DEMPER - 100. Since the maximum realistic value for CLKMIN is 59 and DEMPER is 60, then the result of the above equation is from 0 to 19. Therefore, a 40 added to this result will yield a value between 40 and 59 min, which is the new value of FRZMIN saved at NEW6.

Entry at NEW6 indicates that the hour counter must also be incremented, which in turn may require that the day counter (CLKDAY) also be updated.

When all updates are done, control transfers to NEW99 and the calling program receives control.

PROGRAM NAME - CHKPUL

BRIEF DESCRIPTION - Checks the last pulse time of each input to determine if the maximum time between pulse constraints has been violated.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Wait list recall

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

EXIT MODE - JMP RINO

SUBROUTINES CALLED - CHKP99, BCDBIN, BCDASC, SSP, ALAR1

DYNAMIC VARIABLES USED - WLPUL, CHKPX, CLKMIN, CHKPB, WLTOP+2, CHKPC, BUF5, CHKPA, NINPS

VARIABLES TO BE PRE-INITIALIZED - CHKPX, CLKMIN, CHKPB, CHKPC, CHKPA, NINPS

DETAILED DESCRIPTION -

This routine is entered by a wait-list recall. At the very first time, it is entered at CHKPUL to initialize pointers and set conditions so that future wait-list recalls will enter at CHKP0.

On entry at CHKP0 the variable CHKPX contains a pointer into the table INPTBL, which contains a list of pointers to individual input-data blocks. The LDX 0,X instruction fetches the next input-data block address from this table. The time in minutes of the last recorded input pulse is compared to $FE. If it equals $FE, then the meter has been placed out of service or a load drop has occurred; no further processing is done on the meter, and control transfers to CHKP5.

Otherwise, the BCD minutes stored at LPULT are converted to binary by BCDBIN and saved on the stack for later use. The current value of CLKMIN is also converted to binary and saved at CHKPB. LPULT is then subtracted from CLKMIN; if CLKMIN is greater than or equal to LPULT, control transfers to CHKP1 to compare the maximum pulse time to the value CLKMIN-LPULT. If LPULT is greater than CLKMIN, 1 hr has passed since the last pulse time and a correction factor of 60 is added to CLKMIN, after which LPULT is subtracted. This is also followed by the check for a pulse time violation at CHKP1.

If no time violation occurs, control transfers to CHKP5. If there was a time violation, then pending display programs are cleared and the message "PULSE LATE I#" is output to the display. The BCD input number being maintained at CHKPC is converted to ASCII and stored in BUF5 for output to the display. The audible alarm is then sounded and control falls through to CHKP5.

At CHKP5 the input counter is decremented and if still more inputs remain to be checked, control transfers to CHKP9. Otherwise, all pointers and counters are initialized by CHKP99 and control transfers to CHKP6.

At CHKP9 the BCD input counter is incremented by one; the pointer into the input pointer table is updated to point to the next input-data block address. Then at CHKP6, the routine whose entry point is set to CHKP0 is put on the wait list for 10 s and control transfers to RINO.

PROGRAM NAME - CLK

BRIEF DESCRIPTION - Updates the software clock on each NMI interrupt. Also checks whether it is time to check the wait list.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Status of interrupted program

INTERRUPT STATUS - Interrupt mask bit set.

ENTERED VIA - NMI vectored interrupt

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Status of interrupted program restored by RTI or, if time to check wait list, stack is left unchanged.

INTERRUPT STATUS - Set to status of interrupted program or left the same and control transfers to CWL.

EXIT MODE - RTI or JMP CWL

SUBROUTINES CALLED - NEWPER, NEWBIL

DYNAMIC VARIABLES USED - FRZFLG, NMICNT, INIT, CLKPUL, CLKMIN, CLKSEC, CLKHRS, CLKDAY, FRZSEC, FRZMIN, FRZHRS, NUNIT, INIT1, BILDAY

VARIABLES TO BE PRE-INITIALIZED - FRZFLG, NMICNT, INIT, INIT1, CLKPUL, CLKMIN, CLKSEC, CLKHRS, CLKDAY, FRZSEC, FRZMIN, FRZHRS, NUNIT, BILDAY

DETAILED DESCRIPTION -

This routine maintains the software clock by incrementing BCD values in variables CLKSEC, CLKMIN, CLKHRS, and CLKDAY and by decrementing the binary value CLKPUL.

The routine is entered whenever an NMI interrupt occurs by a vectored interrupt whose address is stored at FFFC. On entry the NMICNT is checked to make sure that the last NMI interrupt is not still being processed when it is time for this entry. If it is being processed, control transfers to ERROR. Otherwise, control transfers to CLK20 and the NMICNT is incremented by one. Then the INIT flag is checked to determine if it is time for the freeze period. This flag is set non-zero by the pre-freeze routine PREFRZ, which sets the flag one CLKPUL time before the freeze period is due.

Once all the flag conditions have been checked and found to be non-responsive, the software-clock updating starts. First the number of pulses (CLKPUL) is decremented. If a non-zero value results, it is not time to increment the seconds counter and control transfers to CLK9. If the seconds counter is to be incremented, CLKPER reinitializes CLKPUL to the number of pulses in 1 s and the seconds counter is updated. If the number of seconds does not equal 60, control transfers to CLK1. Otherwise, the updating of minutes, hours, and days proceeds accordingly until CLK1 is reached by a conditional branch or until a new day is reached. If a new day is required, then the day is compared with the next billing date (BILDAY); if they are equal, then control transfers to the new billing routine. Otherwise, control transfers to CLK1.

At CLK1 the routine checks for a possible pre-freeze indication to see whether CLKSEC, CLKMIN, or CLKHRS is equal to FRZSEC, FRZMIN, or FRZHRS. If so, the routine PREFRZ is executed. Otherwise, control transfers to CLK9. It is important to note that, during the first second of the software-clock operation, the CLKPUL counter is shorted by one count; thus, the pre-freeze time will actually be occurring 10 ms before the appointed freeze time.

CLK9 starts the clock termination process by zeroing NMICNT and decrementing the wait list check counter NUNIT. This counter is set equal to INUNIT counts to establish how often the wait list is to be examined with respect to the number of NMI interrupts caused by the hardware clock. Every INUNIT pulses, control transfers to CWL, and the NUNIT counter is reset. At all other times, an RTI instruction is executed.

If it is one clock pulse before the hour, the PREFRZ point is reached. The INIT1 flag and the NMICNT are zeroed so that no meter input data will be monitored after the current cycle of the RIN program and the NMICNT will be initialized for the entry to CLK. The freeze flag, FRZFLG, is then set non-zero so that the next entry to CLK will cause a branch to FREEZE.

FREEZE is entered at the end of the current demand period. The meter data and freeze times are updated by NEWPER. CLKPUL is decremented to reflect the fact that another clock interrupt has occurred but control has not been allowed to trickle through the clock updating portion of the CLK routine. The INIT1 flag is set non-zero to allow meter monitoring to continue and FRZFLG and NMICNT are zeroed for the next entry into CLK. Control is then returned to the interrupted program by an RTI instruction.

PROGRAM NAME - NEWBIL

BRIEF DESCRIPTION - Updates input and output totals for the new billing period and writes a header record to the tape.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Contains status of program interrupted by NMI (NA on NEWB2 entry)

INTERRUPT STATUS - Interrupt mask bit set.

ENTERED VIA - Jump from CLK routine on entry to NEWBIL. Entered by wait-list recall to NEWB2.

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Status of interrupted program if NEWBIL entry; otherwise, no change.

INTERRUPT STATUS - Interrupt mask bit set on NEWBIL entry; not applicable otherwise.

EXIT MODE - NEWBIL jumps to CLK1; NEWB2 jumps to RINO.

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - BILDAY, WLADC+2, WLADC, INPTBL, INPPNT, OUTPT, OUTTBC, NINPS, NOUTS

VARIABLES TO BE PRE-INITIALIZED - BILDAY, WLADC+2, WLADC, INPTBL, OUTTBL, NINPS, NOUTS

DETAILED DESCRIPTION -

This routine gets control from the CLK routine at the start of each billing period. To avoid operating under the cover of an NMI interrupt for any longer than necessary, this routine does only minimal essential work. First, the BILDAY variable is set to zero so that the next freeze time of the current day does not invoke NEWBIL again. Next, the routine NEWB2 is placed on the wait list for 25 s. This is to allow time for the normal freeze-time functions that may coincide with the new billing time functions. Control then transfers to CLK1 to finish processing the clock interrupt.

NEWB2 is entered from the wait list approximately 25 s after the start of the new billing period. The HEADER routine outputs information to the tape to close the tape for the current billing period. Next, a pointer into the table INPTBL is established to point to the addresses of the input-data blocks. This pointer is saved at INPPNT. The number of inputs is maintained in the A register to determine when all inputs have been checked. The addresses for each data block are sequenced in turn so that the total pulse counter in each block (NPULT) can be zeroed. After all inputs are done, control transfers to NEWB4 and all the output totals are zeroed in a similar manner.

When all the outputs are updated the ADC routine is placed back on the wait list for 1 s and control transfers to RINO.

PROGRAM NAME - CLRMEM

BRIEF DESCRIPTION - Initializes a specified block of RAM memory to a specified initial value.

ON ENTRY -

A REGISTER - Initialization value B REGISTER - NA

X REGISTER - First location to be initialized

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - Unchanged

X REGISTER - NA

STACK - Unchanged

INTERRUPT STATUS - Unchanged

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - CLRTOP

VARIABLES TO BE PRE-INITIALIZED - CLRTOP

DETAILED DESCRIPTION -

This routine is entered with the X register equal to the first address to be initialized. The value in the A register is stored in this location and all subsequent locations up to, but not including, the address stored at CLRTOP. Note that CLRTOP is at the lowest location in RAM so it will not be destroyed during the initialization operation.

PROGRAM NAME - CWL

BRIEF DESCRIPTION - Checks the wait list to determine whether any programs are due for recall.

ON ENTRY -

A REGISTER - NA　　　B REGISTER - NA

X REGISTER - NA

STACK - Contains status of program interrupted by NMI.

INTERRUPT STATUS - Interrupt mask bit set.

ENTERED VIA - Entered from CLK by JMP CWL.

ON EXIT -

A REGISTER - NA　　　B REGISTER - NA

X REGISTER - NA

STACK - Status of interrupted program returned on exit

INTERRUPT STATUS - Restored to status of interrupted program unless recalled

EXIT MODE - RTI or JMP XQT

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - WLPNT, WLTOP, CWLX

VARIABLES TO BE PRE-INITIALIZED - WLPNT, WLTOP

DETAILED DESCRIPTION -

This routine gets control from CLK every INUNIT hardware clock pulse. To make sure that multiple users will not call XQT while it has been interrupted in processing another address, XQTFLG, a flag which is controlled by XQT, is checked. If it is zero, XQT is not busy and control transfers to the LDX WLPNT instruction. Otherwise, control transfers to CWL99 where an RTI instruction is executed. WLPNT contains a pointer into the wait-list entries which are each 3 bytes long. They contain a program address high-order byte, a low-order address byte, and a byte counter. The top portion of the wait list has entry areas reserved for specific programs or functions. The wait-list area starting at WLFREE is not reserved and can be used by any program. This area is usually allocated by the SSWAIT or WAIT routines. The wait list is terminated by a 0 byte in the high-order address byte of an entry.

If the pointer points to the end of the list, on entry, control will transfer to CWL7. Otherwise, the wait time left for the entry being examined is checked. If it is zero, the corresponding entry is not waiting. If it is non-zero, it is then decremented by one count and again examined for a zero condition. If it is now zero, it is time to recall the program specified in the address portion of the entry and control transfers to CWL9.

If an entry is not due for recall, the next sequential entry is checked until the end of the list is found or until it is time to recall a program. In either case, all entries will not be checked on each pass through CWL.

At CWL7 the WLPNT is reinitialized to point to the top of the list, and control transfers to CWL1. The pointer is also reset at CWL8, but no further processing is done since at least one wait-list entry was checked.

On entry at CWL9, it has been determined that it is time to recall a program from the list. The X register points into the wait list at the address of the program to be recalled. Therefore, the LDX 0,X instruction retrieves the program-recall address. The WLPNT pointer is then updated to point to the next successive wait-list pointer and the recalled program address is placed on the FIFO stack by XQT to await execution.

PROGRAM NAME - MSGPRO and MSGKEY

BRIEF DESCRIPTION - Processes messages input by keypad or Teletype into the system and transfers control to the proper handler.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Entered from FIFO stack processor.

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - NA

INTERRUPT STATUS - NA

EXIT MODE - RTS or JMP XQTS

SUBROUTINES CALLED - TRY

DYNAMIC VARIABLES USED - BUF1, TINPNT, MSGTOP, WLTOP+2, KEYPNT

VARIABLES TO BE PRE-INITIALIZED - MSGTOP (table)

DETAILED DESCRIPTION -

This routine has two entry points: MSGPRO and MSGKEY. MSGKEY is entered from MSGATN after a keyboard attention interrupt has been generated at the keypad. MSGPRO is entered when the return key is depressed on the Teletype. Both entries transfer control to MSG1 with the address of the input buffer in the X register.

At MSG1 the first ASCII character from the input buffer is loaded into the A register and the second character, into the B. Any pending display programs are removed from the wait list, the ASCII characters are converted to BCD by ASCBCD, and the result in the A register is compared to an op-code from the message process table in the B register. If the two-digit input matches the op-code, then control transfers to MSG8. Otherwise, the pointer into the message process table is updated and then a check is made to see if a $FF terminator is found. If the terminator is found, then the input op-code was invalid, the TRY routine is called, and control passes to RINO. If a terminator was not found, the program loops to MSG2 and checks the rest of the op-codes until it finds a match or terminator.

MSG8 is reached if an op-code is found. The corresponding address is retrieved from the table and placed on the FIFO stack by XQTS to await execution. Control then transfers to RINO.

PROGRAM NAME - BCDASC and BCDAS1

BRIEF DESCRIPTION - Converts two packed BCD numbers to two unpacked ASCII characters.

ON ENTRY -

A REGISTER - Two packed BCD numbers   B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - First ASCII character   B REGISTER - Second ASCII character

X REGISTER - Unchanged

STACK - NA

INTERRUPT STATUS - Unchanged

EXIT MODE - RTS

SUBROUTINES CALLED - BCDAS1 calls BCDASC

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

On entry the A register contains two packed BCD characters with the most significant half byte to the left and the least significant half byte to the right. The A-register value is transferred to B for safe keeping and the most significant half byte is shifted to the right half byte position.

ORing a $30 with A produces the proper corresponding ASCII representation of the number. The B register least significant value is then fixed by masking off the most significant half byte and ORing a $30 with B. Control returns to the calling program.

The entry point BCDAS1 is used when it is desired to store the ASCII results in a buffer pointed to by the X register. BCDASC is called to convert the data to ASCII, the characters are stored in the specified buffer, and the buffer pointer is incremented by two.

PROGRAM NAME - GETDTA

BRIEF DESCRIPTION - Outputs a specified message to the display and inputs a designated number of characters in response to the message.

ON ENTRY -

A REGISTER - Number of characters to input   B REGISTER - NA

X REGISTER - Address of message to be output

STACK - NA

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA   B REGISTER - NA

X REGISTER - Address of where input data are stored

STACK - NA (used for temporary storage, but returned to same status as on input)

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - INP, BLANKS, SSP16

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

On entry to this routine, the A register contains the number of characters to be input in response to the message pointed to by the X register. The input count is saved on the stack for subsequent recall as a parameter for the transfer to INP. The routine BLANKS outputs as many blanks as there are input character responses. This guarantees that extraneous characters stored in the non-displaying buffer of the display will not be shifted to the visible buffer during the backspace operation which accompanies the display of characters in place of "?". SSP16 is then called to display the message whose address is in the X register. Finally, control transfers to the INP routine to input the response. The INP routine will return to the caller of GETDTA with the X register pointing to the input buffer where the ASCII response will be found.

PROGRAM NAME - POWERT and POW1

BRIEF DESCRIPTION - Displays the messages for power-type input and returns the operator response to the calling program.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - BCD operator response   B REGISTER - NA

X REGISTER - Address of data block currently being initialized

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - POW1 called by POWERT, SSP, SSWAIT, GETDTA

DYNAMIC VARIABLES USED - POWX, OUTTOP

VARIABLES TO BE PRE-INITIALIZED - OUTTOP

DETAILED DESCRIPTION -

Since both IID and IOD need to input the power type, a single routine was written to cut down on storage requirements. This routine first calls POW1, which outputs the five-character message in BUF5. If the call was from IID, the message is "I#XX". If the call was from IOD, then the message is "O#XX". The value of XX is the current input or output number being initialized. After this preamble, the message "POWER TYPE" is output. Therefore, the display will show a message of the form "I#08 POWER TYPE". This message is displayed for about 2 s and then POW1 returns to the calling program.

POWERT then displays the message "KW=1, KVAR=2: A=?" by GETDTA which also inputs the one-digit response. The ASCII response is converted to BCD by masking off the high-order four bits. The address of the current data block being initialized is at OUTTOP and INTOP since they are equivalent, and the BCD response is then returned to the caller in the A register.

PROGRAM NAME - USE1

BRIEF DESCRIPTION - Utility program to increment two values and return the results in the A and X registers.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - Input count +1 B REGISTER - NA

X REGISTER - INPPNT value +2

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - INPPNT, STATA

VARIABLES TO BE PRE-INITIALIZED - INPPNT, STATA

DETAILED DESCRIPTION -

This routine is a utility program used by the STATUS routine and its subroutines. It fetches the input count from STATA and adds one to it. The current value of INPPNT is also incremented by two and placed in the X register. Control then returns to the calling program.

PROGRAM NAME - USE3

BRIEF DESCRIPTION - Displays a message and inputs a four-character ASCII response which is converted to BCD and returned to the caller.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - Address of message to be displayed

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - Low-order BCD response   B REGISTER - High-order BCD response

X REGISTER - Points to buffer containing ASCII data.

STACK - PSH and PUL executed during program execution. Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - GETDTA, ASCBCD

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This is a utility routine which displays the message whose address is in the X register and inputs a four-digit response via GETDTA. The four-digit ASCII response is converted to BCD by successive calls to ASCBCD. On exit, the high-order digits are in the B register and the low-order digits are in the A.

PROGRAM NAME - SCALEF

BRIEF DESCRIPTION - Outputs a scale-factor request message and inputs the operator's response.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - High-order scale   B REGISTER - Low-order scale

X REGISTER - Address of data block being initialized

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - INPDTA, ASCBCD, CNV1

DYNAMIC VARIABLES USED - INPTOP

VARIABLES TO BE PRE-INITIALIZED - BUF5 (used by INPDTA)

DETAILED DESCRIPTION -

SCALEF is called by IID or IOD to input the scale factor for a specified input or output. The routine INPDTA outputs a message in the form "I#01 SCALE=????" or "O#03 SCALE=????" depending on the caller. The four-character response is input in ASCII and converted to BCD by successive calls to ASCBCD. The routine CNV1 is then called with the high-order BCD scale factor in the A register and the low order in the B. CNV1 returns a binary result at Q. However, since the result cannot exceed 9999 only the low-order 2 bytes of Q are used. The X register is set equal to the data block being processed, the high-order binary scale factor is returned in the A register, and the low order, in the B.

PROGRAM NAME - MNUMBS

BRIEF DESCRIPTION - Displays a request for the number of inputs or outputs and inputs the response.

ON ENTRY -

A REGISTER - NA | B REGISTER - NA

X REGISTER - Address of message to output

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - Binary number of inputs or outputs | B REGISTER - NA

X REGISTER - Address where ASCII response is stored

STACK - Return address pulled from stack by BCDBIN return.

INTERRUPT STATUS - NA

EXIT MODE - JMP BCDBIN

SUBROUTINES CALLED - GETDTA, ASCBCD, BCDBIN

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

MNUMBS inputs the number of inputs or outputs (depending on the caller). The routine is entered with the X register pointing to the message to be displayed. GETDTA displays the message "METER INPUTS=??" or "METER OUTPUTS=??" and inputs the two-digit response. These ASCII digits are converted to BCD by ASCBCD and then converted to binary by BCDBIN, which also returns control to the caller of MNUMBS.

PROGRAM NAME - ASCBCD

BRIEF DESCRIPTION - Converts two ASCII characters to packed BCD.

ON ENTRY -

A REGISTER - First ASCII character   B REGISTER - Second ASCII character

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - Packed BCD characters   B REGISTER - NA

X REGISTER - Unchanged

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

On, entry, the A register contains the high-order ASCII character to convert to BCD and the B contains the low order. The four low-order bits of A are shifted to the left half byte and the right half byte is set to zeroes. The four high-order bits of B are masked out and the two half bytes are merged together by the ABA instruction.

PROGRAM NAME - BCDBIN

BRIEF DESCRIPTION - Converts two packed BCD characters to their binary equivalent.

ON ENTRY -

A REGISTER - Two packed BCD characters B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - Binary equivalent B REGISTER - NA

X REGISTER - Unchanged

STACK - Return address pulled from stack on exit (data saved temporarily on stack but returned to status of the entry).

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine uses the add/shift multiplication technique to multiply the tens BCD digit by 10 and add to this the units BCD digit. Since a

SUBROUTINES CALLED - HEXASC called by HEXAS1

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

On entry two packed hex characters are saved in the A and B registers. The right half byte is converted to ASCII by HEX5 and pushed onto the stack temporarily. The original left half is then shifted to the right half byte and also converted to ASCII by HEX5. On return from HEX5, the A register equals the ASCII code for the original most significant hex character.

HEX5 checks whether the value in the A register is less than, equal to, or greater than 9. If A is from 0 to 9, a $30 is OR-ed with the A register to form an ASCII digit. Otherwise, the character is a hex A-F and 9 is subtracted from the A register so the new value in the A will be from 0 to 5; a $40 OR-ed with this will yield an ASCII character from "A" to "F".

HEXAS1 is an alternate entry which is used when it is desired to store the ASCII results in a buffer pointed to by the X register. HEXASC is called to convert the data to ASCII; the characters are then stored in the specified buffer and the buffer pointer is incremented by two.

PROGRAM NAME - CNV1

BRIEF DESCRIPTION - Converts four packed BCD characters into binary.

ON ENTRY -

A REGISTER - Low-order BCD B REGISTER - High-order BCD

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

binary 10 is represented as "1010" this requires a shift, add/shift, shift, and add/shift sequence to perform the multiplication. The high-order digit is worked on first. It is in the upper four bits of the A register and the lower four bits are masked off. The first shift is then performed because of the low-order "0" multiplier. Next, since the product is currently zero, no real addition needs to take place so the product, in the A register, is temporarily saved on the stack. A shift associated with this pseudo-addition to zero is followed by a shift to complete a multiplication by 1. The next shift is performed because the third bit of the multiplier is a zero. The product from the stack is then added to the multiplicand to form the new product in the A register. The original BCD value to be converted is then pulled from the stack and the high-order bits are masked off to leave only the unit bits left. This value is added to the product of the high-order BCD digit × 10, and the result is returned in the A register.

PROGRAM NAME - HEXASC and HEXAS1

BRIEF DESCRIPTION - Converts two packed hex characters into ASCII characters.

ON ENTRY -

A REGISTER - Two packed BCD hex characters

B REGISTER - NA

X REGISTER - On entry to HEXAS1, X equals address where ASCII characters are to be saved

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - High-order ASCII character

B REGISTER - Low-order ASCII character

X REGISTER - Address of next available buffer location if HEXAS1 entry

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

X REGISTER - NA

STACK - Return address pulled from stack on exit (used for temporary storage but returned to status of the entry).

INTERRUPT STATUS - NA

EXIT MODE - RTS (results in Q)

SUBROUTINES CALLED - BCDBIN, MULTL

DYNAMIC VARIABLES USED - Q, XA, YA

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

On entry, the B register equals the high-order BCD digits to be converted to binary and the A contains the low-order digits. The low-order digits are first converted to binary by the BCDBIN routine. The result is saved in the low-order product byte used by the MULTL routine. BCDBIN converts the high-order digits to their binary equivalents, and effectively divides them by 100 by storing them in the least significant byte of the multiplier location for MULTL. A multiplicand of 100 is then saved for use by MULTL. The result computed by MULTL is 100 × (high-order binary) + (low-order binary) = result. The result will be stored at Q+2 and Q+3 and MULTL returns control to the caller.

PROGRAM NAME - INPDTA

BRIEF DESCRIPTION - Outputs message to self-scan display starting at BUF5 followed by message whose address is in X. After the message is output, a designated number of characters is input.

ON ENTRY -

A REGISTER - Number of characters to input   B Register - NA

X REGISTER - Address of second part of output message

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - Address of input buffer

STACK - Return address pulled from stack on exit (stack used for temporary storage but returned to status of the entry).

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - SSP, INP, BLANKS

DYNAMIC VARIABLES USED - BUF5, INPX

VARIABLES TO BE PRE-INITIALIZED - BUF5

DETAILED DESCRIPTION -

This routine displays 5 characters from BUF5 followed by 11 characters from a location specified on entry by the X register. The output message requests a response from the operator and the A register equals the number of digits (K) in the response. K-blanks are then output to the display so that the nonvisible storage area of the display will be flushed sufficiently so that the K-backspaces caused by the INP routine will not cause extraneous characters to appear on the left of the display. The blanks are followed by the 5 ASCII characters stored in BUF5, followed in turn by 11 characters stored at the address which is saved at INPX. The number of expected responses is then retrieved from the stack and INP is called to input the operator's response.

PROGRAM NAME - SSWAIT and WAIT

BRIEF DESCRIPTION - Puts the calling program on wait list for a specified time and transfers control to RINO.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT

A REGISTER - NA　　　　B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack before exit unless no space in wait list.

INTERRUPT STATUS - NA

EXIT MODE - RTS if wait list full; otherwise, JMP to RINO

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - WLFREE, BLKTIM

VARIABLES TO BE PRE-INITIALIZED - BLKTIM

DETAILED DESCRIPTION -

This routine, more than any other, has the power to bless or curse the operation of the system. It must not be used indiscriminately for it can cause widespread havoc. However, it must be used in certain cases or the system would be locked into a long programming loop which would preclude acquisition of meter input data.

The entry points SSWAIT and WAIT are different only in that the former uses a preselected wait time stored at BLKTIM and the latter has the wait time specified in the A register. The address of the free-usage portion of the wait list is placed in the X register and the times of succeeding entries on the wait list are examined until a zero wait time is found. This indicates a free entry; the calling program return address is pulled from the stack and saved in the corresponding wait-list entry along with the wait time specified in the A register. If no free slots exist, the caller gets control by an RTS, thereby "pretending" that the wait time has elapsed. This full wait-list event is inserted to prevent a "compute-bound" situation which might otherwise occur.

Aside from a full wait-list problem there is also the possibility of a misplaced-subroutine linkage return from the stack. This problem has been taken care of in the current version of the system but the reader should be aware of the problem for future system expansion. The problem occurs when two programs, P and Q, which are "called-subroutines," both find themselves on the wait list at the same time. If P was placed on the wait list first, the return address of its calling program will be closer to the bottom of the stack than Q's calling program, provided that Q was placed on the stack after P. If Q is on the stack for a longer time than P, then P will get control before Q, and when it executes an RTS it will return to Q's caller instead of its own.

One way around this problem is to arrange for programs that use the wait list to save their calling program's return in a special location and not on the stack. In other cases, the misdirected return is okay as long as all return paths are eventually executed.

PROGRAM NAME - INP

BRIEF DESCRIPTION - Inputs a specified number of characters from key-pad and displays them on self-scan display with requisite number of backspaces.

ON ENTRY -

A REGISTER - Number of input characters

B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA

B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - SSBKSP, IWAIT, SSP, SSCHAR

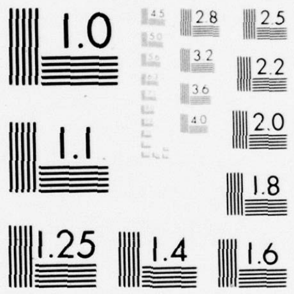
DYNAMIC VARIABLES USED - INPA, KEYCNT, KEYPNT, BUF4, INPB, INPX1

VARIABLES TO BE PRE-INITIALIZED - None

AD-A042 562 HARRY DIAMOND LABS ADELPHI MD F/G 9/2
ARMS REMOTE PROGRAMMER'S MANUAL.(U)
JUL 77 D J BÜSCHER, M A RESSLER
UNCLASSIFIED HDL-SR-77-1 NL

2 OF 2
ADA042562

END
DATE
FILMED
8-77
DDC


1.0
4.5
5.0
5.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

DETAILED DESCRIPTION -

On entry, the A register equals the number of inputs expected. This value is stored at INPA, INPB, and KEYCNT, respectively. The caller's return address is then pulled from the stack and saved at INPX1 and the input buffer pointer into BUF4 is initialized and stored at KEYPNT.

At INP1 the routine enters a loop to await the depression of a keypad key. Since the KEYIRQ routine decrements KEYCNT on each entry, a comparison between KEYCNT and INPB will yield a non-zero result only if a key has been depressed. Therefore, on a zero result the INP1 loop is continued. Otherwise, the display is backspaced by the number of characters expected for input. The pointer to the character input is then established in the X register and the character is output to the display by SSP. The number of inputs expected is then reduced by one and KEYCNT is examined to determine if all inputs have been acquired. If all characters are done, control transfers to INP2. If not, a question mark is output for each input that remains. Therefore, as the operator types a response, successive question marks are replaced from left to right by the response. The INP1 loop is then repeated until all inputs have been handled.

At INP2 the key counter is set to 1 to await the proper terminating key response. The INP3 loop is entered and continues until a key is depressed. The character input is fetched from the buffer pointed to by KEYPNT and compared to all possible control codes. If a match is found, a corresponding routine gets control. If there is no match with a control code, it is assumed that the operator is not satisfied with his input and wishes to replace it with another number. In this case the display must be made to look as it did after only one input had been handled. First the original number of inputs is fetched from INPA and the display is back-spaced that many characters. Then the input character just handled is output to the display. The variables INPB and KEYCNT are then set equal to the number of inputs expected minus one. Control then transfers to INP12 to output the requisite number of question marks and await further input.

INP4, INP6, INP7, and INP8 are for "EXECUTE," "PERIOD," "ATTENTION," and "SPACE" key hits, respectively. In this version, all these responses translate into an EXECUTE response where the buffer is terminated with a zero and the X register is returned to the caller containing the address of the input buffer.

If a comma key was depressed, then the comma is effectively removed from the buffer by setting the value in KEYPNT to point to the comma entry. Therefore, the next input will overwrite the comma. The comma, however, is output to the display and the variables KEYCNT and INPB are set back to the initial value. Control then returns to INP12 for question-mark output and subsequent input.

PROGRAM NAME - OP00

BRIEF DESCRIPTION - This routine handles operator requests for the display of input meter data.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - GETDTA, ASCBCD, BCDBIN, OP00U1, BCDAS1, BCDAS2, SSP16, BCDMLT, BCDASC, USE2

DYNAMIC VARIABLES USED - NINPS, INPTBL, OP00X1. OP00A, INPPNT, CLKMIN, CLKHRS, CLKDAY, INPTOP, OP00X, WLTOP+2, WLTOP, YA, XA, Q

VARIABLES TO BE PRE-INITIALIZED - NINPS, INPTBL, CLKMIN, CLKHRS, CLKDAY

DETAILED DESCRIPTION -

This routine is invoked by the message processor in response to a "00" op-code input. The first message displayed ("WHICH INPUT ??") requests the operator to indicate which input's data he would like displayed. A two-character response is input by GETDTA and this response is stored in ASCII form in BUF5 and converted to BCD and then to binary by ASCBCD and BCDBIN, respectively. If the input number requested is beyond the allowable range, control is returned to OP00. Spaces are then stored in BUF5 to guarantee that displays of only four numbers will be followed by blanks. Displays of six numbers plus accompanying preamble information will overwrite these blanks. The ASCII characters "I#" are also stored in the buffer for subsequent display.

OP00U1 is then entered with the X register pointing to a table of input- or output-data-block addresses and the A register contains the input or output number request. The reason that inputs and outputs are both allowed is that this particular code is used both by OP00 and OP01. The proper index into the specified pointer table is determined by the equation

effective table address = (input number - 1) × 2 + first word address of table.

This effective address is stored at INPPNT (also, same as OUTPT) and the actual pointer (pointed to by the contents of INPPNT) is then loaded into the X register and saved at INPTOP. Control then returns to the caller.

Next the address of the table containing valid jump addresses for types of input data requests is saved at OP00X1, the maximum number for such a request is stored at OP00A, and control transfers to OP00U2.

At OP00U2, the message "WHICH DATA ??" is output and a two-character response is input by GETDTA. This response is converted into binary and compared with the maximum number allowed. If an illegal request has been made, control transfers to OP00U2; otherwise, the proper pointer into the jump address table is computed as it was for the input or output pointer table and the resulting jump address is saved at OP00X. This address is then pushed onto the stack and the X register is loaded with the address of the requested input or output-data block. An RTS instruction will then cause control to transfer to the requested routine.

At OP0000 through OP0007, the requested value is loaded into the A and B registers and then control transfers to OP000, OP001, OP002, OP003, or OP004.

OP000 is entered if the requested data are pulse-type data and are to be updated on the display at a 0.3-s interval. Control then transfers to OP001.

At OP001, pulse-type data are converted to ASCII and stored in the output buffer along with the "PUL=" message and the previously stored input-number preamble. Then a message of the form "I#02 PUL=1742" is displayed. Control then transfers to OP0099.

OP002 is an alternative entry point to OP001 that is entered when six-digit numbers are being processed.

OP003 is used to compute KWH or KVARH values given the pulse count. OP003 is entered for two-byte conversions and OP004 is entered for three-byte conversions, the difference being that either YA's high-order byte is cleared or pre-initialized. YA is then further initialized to contain a BCD pulse count which will be the multiplicand for the BCDMLT routine. The multiplier XA is set to the binary value of the input scale factor. The routine BCDMLT then performs the binary/BCD multiplication and the results in BCD are returned in the six-byte Q table. The seven low-order BCD digits of the result at Q are converted to ASCII by BCDASC or BCDAS1 and the results are saved in the output buffer. The power type is then determined from the input-data block (PWRTYP) location.

At OP005, the X register will contain the address of DMSG21 or DMSG42, if the power type was KWH or KVARH, respectively. The "KWH=" or "KQH=" message will then be placed in its proper place in the output buffer. The resulting message output by SSP16 will be of the form "I#04 KWH=0123456".

If a load drop/resume or service start/stop request was entered, then one of the entry points OP0008 through OP0011 will be invoked. At OP008 a $FE byte will be saved at the LPULT location of the requested data-input block. This indicates to other routines that this input has requested a load drop. The last address of the message "DROPPING LOAD" is then placed in the X register and control transfers to OP0081.

OP0081 and OP0082 are points that are entered from OP0009, OP0010, and OP0011. The entry point OP0082 is used by OP0010, since it stores meaningful data at BUF2+19 and BUF2+20. The rest of this routine, starting at OP0083, fetches the contents of the buffer, referenced by the X register, and stores 16 characters on the stack. The top of the stack will finally contain the first character to be displayed. When 16 characters have been transferred to the stack, the X register will point to one location before the message in the original buffer. The X register is incremented to point to the first character and SSP16 is then called to display the message. The stack is then unloaded in the OP0084 loop and the data are placed in BUF2 for output to the Teletype. The output is completed by adding the input number, some spaces, a date/time stamp, a carriage return, and a line feed.

The Teletype-output recall address is then set to RINO and the output buffer address is passed to USE2 to initialize the Teletype-output sequence.

At OP0009, the resume-load request causes the current CLKMIN value to be stored in LPULT location of the current input-data block. This indicates to other programs that the particular meter input is back in service. Control then transfers to OP0081.

At OP0010, the current input is placed out of service by setting its LPULT location to $FE. Then the message "REPLACED BY ??" is output and a two-character response is input by GETDTA. The response is saved in the output buffer and processing continues at OP0082.

At OP0011, the current input is placed back in service by setting its LPULT location to the current CLKMIN value. Control is then transferred to OP0081.

PROGRAM NAME - OP01

BRIEF DESCRIPTION - This routine handles operator requests for the display of output meter data.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - GETDTA, ASCBCD, BCDBIN

DYNAMIC VARIABLES USED - NOUTS, OUTTBL, OP01A, YA, XA

VARIABLES TO BE PRE-INITIALIZED - NOUTS, OUTTBL

DETAILED DESCRIPTION -

The processing for OP01 is virtually identical to that of OP00 except that variable names may change and there are no data requests past OP0107. Otherwise, the reader can gain an understanding of the OP01 routine by reading the documentation for OP00.

PROGRAM NAME - OP02

BRIEF DESCRIPTION - Displays the instantaneous KW value on the display. This value is updated every 13 s.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - SSP, BCDMLT, BCDAS1

DYNAMIC VARIABLES USED - OP02A, OP02X, OP02X1, KWSCL, XA, YA, Q, WLTOP, WLTOP+2, KWDTA

VARIABLES TO BE PRE-INITIALIZED - KWSCL, KWDTA

DETAILED DESCRIPTION -

The first part of this routine gets the last instantaneous KW reading from KWDTA and saves it at OP02A. The message address for "KW=" is then stored at OP02X and the KWSCL value is placed in the A and B registers.

OP021 is a common continuation point for both OP02 and OP03. First the BCD KWSCL or KVASCL value is stored for use by BCDMLT in the two low-order bytes of the multiplicand register. Next the high-order multiplier and multiplicand bytes are zeroed. An ASCII "plus" or "minus" sign is then stored in the display-output buffer, depending on the value of the last KW or KVAR reading. If the reading was from 128 to 255, then the analog output current was from zero to plus full scale. This makes the ADC reading a sign-magnitude type of representation of the instantaneous KW or KVAR reading. Therefore, the most significant bit of the reading is stripped off and the absolute value of the reading remains. This value is stored as the low-order multiplier byte for BCDMLT.

The message "NOW" is then output to the display followed by the message "KW=" or "KVAR=". BCDMLT is then called and the product is returned in BCD in the three low-order bytes of Q. This value is then converted into ASCII and placed in the output buffer BUF5 by successive calls to BCDAS1. When the buffer is done it is output to display a message of the form "NOW KW=+123456".

The return address is then fetched from OP02X1 and put on the wait list for about 13 s. Control then transfers to RINO.

PROGRAM NAME - OP03

BRIEF DESCRIPTION - Displays the instantaneous KVAR value on the display. This value is updated every 13 s.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - KVADTA, OP02A, OP02X, OP02X1, KVASCL

VARIABLES TO BE PRE-INITIALIZED - KVADTA, KVASCL

DETAILED DESCRIPTION -

This routine works the same as OP02 except that KVAR values are substituted for KW values.

PROGRAM NAME - OP41

BRIEF DESCRIPTION - Allows the operator to restart the program at the end of the data-initialization phase.

ON ENTRY- NA

ON EXIT - NA

SUBROUTINES CALLED - PASWRD

DYNAMIC VARIABLES USED - INIT, INIT1

VARIABLES TO BE PREINITIALIZED - None

DETAILED DESCRIPTION -

This program is primarily for debugging purposes and it allows the operator to restart the program at IOD9. When the proper password is entered, the INIT and INIT1 flags will be cleared to temporarily suspend the software clock and data-gathering operations.

PROGRAM NAME - OP50

BRIEF DESCRIPTION - Displays the current time on the self-scan display.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - BCDASC, SSP

DYNAMIC VARIABLES USED - WLTOP, BUF3, CLKHRS, CLKMIN, CLKSEC, SSPFLG

VARIABLES TO BE PREINITIALIZED - WLTOP, CLKHRS, CLKMIN, CLKSEC

DETAILED DESCRIPTION -

This routine is first entered at OP50 but subsequent recalls from the wait list will enter at OP501. At OP50, SSPFLG is cleared to indicate that all buffers used by the rest of the program have not been initialized. Subsequent entries at OP501 will see SSPFLG set to indicate that the buffers have been initialized.

The first part of the routine converts hours, minutes, and seconds to ASCII and saves the results in BUF3. Then the least significant second digit is checked to see if it has changed since the last display time. If not, there is no need to update the display and control transfers to OP506. If SSPFLG is non-zero, then the rest of the buffer is initialized and control transfers to OP505. If updating is required, then the buffer is completed with spaces and colons and then displayed in the form "12:02:37".

The program is then placed on the wait list for 1 s and control transfers to RINO.

PROGRAM NAME - OP51

BRIEF DESCRIPTION - Displays the current Julian date.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - SSP, BCDASC

DYNAMIC VARIABLES USED - CLKDAY, BUF5

VARIABLES TO BE PRE-INITIALIZED - CLKDAY

DETAILED DESCRIPTION -

This routine outputs the message "TODAY'S DATE=" and then converts the Julian date counter from BCD to ASCII. The result is stored in BUF5 and output so that the resulting message is displayed in the form "TODAY'S DATE=123". Control then transfers to RINO.

PROGRAM NAME - OP60

BRIEF DESCRIPTION - Notifies the system that the operator is removing the tape from the drive.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - SSP16, TAPCLO, REWIND

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine allows the user to remove the tape from the drive in an orderly fashion. First the message "**WORKING**" is displayed. Then the tape file is closed and the tape is rewound. When the tape is done, the "REMOVE TAPE NOW" message is displayed and control transfers to RINO.

PROGRAM NAME - OP61

BRIEF DESCRIPTION - Notifies the system that a tape is being inserted in the drive.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - SSP16, REWIND, LOADP, CHKTAP, GETDTA, TAPEIN, TPBKSP, HEADER

DYNAMIC VARIABLES USED - BUF2

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine notifies the system that a tape is to be loaded into the tape drive. The message "PLEASE LOAD TAPE" is displayed and the tape is then rewound, moved to the load point, and checked to make sure it is in record mode. If not, the OP61 loop is reexecuted until the tape is ready.

When the tape is ready at the load point, the message "NEW=1, OLD=2: A=?" is displayed and a one-digit response is input by GETDTA. If the response is a 1, indicating a new tape, then it is assumed that the tape has no useful information. Therefore, a header record is written on the tape and control transfers to RINO.

If it is an old tape, it contains useful information. Therefore, at OP612, tape records are input until a record is found which starts with a $FF byte. This indicates an end-of-file (EOF) condition and the tape is backspaced over this record. The tape is now ready so that data can be appended to the existing tape.

PROGRAM NAME - OP70

BRIEF DESCRIPTION - Requests the operator to input the date of the start of the next billing cycle.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - GETDTA, ASCBCD

DYNAMIC VARIABLES USED - BILDAY

VARIABLES TO BE PRE-INITIALIZED - BILDAY

DETAILED DESCRIPTION -

This routine should be executed once during each billing period. This will normally be a master-station function, but the operator will be required to supply the information during the prototype evaluation.

The routine displays the message "NEXT BILLING=??" and inputs a three-digit response by GETDTA. The response is converted to BCD by successive calls to ASCBCD and the results are saved in BILDAY. The BILDAY variable is checked daily by the CLK routine to determine whether it is time to start a new billing cycle.

PROGRAM NAME - OP80

BRIEF DESCRIPTION - Clears the display to all blanks.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - SSCLR

DYNAMIC VARIABLES USED - WLTOP+2

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine simply clears the display by removing any pending display programs from the wait list and calling the routine SSCLR.

PROGRAM NAME - OP81

BRIEF DESCRIPTION - Turns off the audible alarm.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PREINITIALIZED - None

DETAILED DESCRIPTION -

This routine sets the bit controlling the audible alarm to a 1. This causes the alarm to turn off if it is on. Otherwise, there is no effect. Control then returns to RINO.

PROGRAM NAME - OP82

BRIEF DESCRIPTION - Turns on the audible alarm for 5 s.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - OP81, WAIT

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PREINITIALIZED - None

DETAILED DESCRIPTION -

This routine tests the audible alarm by setting the audible alarm bit to zero for about 5 s and then branching to OP80 to turn the alarm off.

PROGRAM NAME - OP90

BRIEF DESCRIPTION - Dumps the contents of RAM to the Teletype.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - HEXASC

DYNAMIC VARIABLES USED - OP90X1, OP90X, OP90A, TOTRCL, TOTPNT

VARIABLES TO BE PREINITIALIZED - None

DETAILED DESCRIPTION -

This routine is primarily a debugging routine that allows the programmer to examine the contents of RAM.

The pointer into RAM is initialized to zero and saved in OP90X1. The contents of OP90X1 are then loaded into the A register and converted from a half-byte hex representation to ASCII. The same is done to the low-order memory address. All four ASCII address characters and some space characters are stored in BUF2 for later output. The next available output buffer location is stored at OP90X and then the next 16 locations starting at the current value of OP90X1 are fetched and converted to ASCII and saved in the buffer. Each two-character sequence, which represents one byte of memory, is separated by a blank character and stored in the next available location in BUF2. When 16 locations are done, as determined by the counter maintained at OP90A, control drops through to OP904. The last location in the buffer is set to zero as a terminator for the Teletype output routine and the recall address is set to OP905. A carriage return and line feed are stored at the beginning of BUF2 and the buffer address is saved at TOTPNT.

At USE2, a routine used by other programs, the Teletype-output interrupt is allowed (under cover of a set interrupt mask bit) and control transfers to RINO.

When the buffer has been output, control returns to OP905 and the X register is checked to determine if all of RAM has been checked. If so, a buffer with a final carriage return and line feed is set up and a recall address of RINO is established. Control transfers to OP904 to output the final buffer.

If more data are left, the routine loops to OP901 to output the next line.

PROGRAM NAME - OP91

BRIEF DESCRIPTION - Outputs the tape to the Teletype.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - GETDTA, REWIND, EOF, LOADP, TAPEIN, TAPASC

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

The message "NEW=1, OLD=2: A=?" is displayed and the one-digit response is input by GETDTA. If a 1 was input, this indicates that the tape to be printed is "new" to the system. That is, it is not the tape that the system is currently using to record data. In this case, the tape is assumed to already have an EOF terminator and control transfers to OP910 to rewind the tape and move it to the load point.

At OP9111 one tape record is read by TAPEIN, converted to ASCII, and output to the Teletype by TAPASC. The OP9111 loop is continued until the TAPASC routine recognizes an EOF and transfers control to RINO.

If an old tape was selected, the tape must be marked with an EOF so the TAPASC routine will have a terminator to look for.

PROGRAM NAME - OP93

BRIEF DESCRIPTION - Outputs the last N number of records from the tape to the Teletype. N is user specified between 0 and 99.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - GETDTA, ASCBCD, BCDBIN, TPBKSP

DYNAMIC VARIABLES USED - OP93A

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

An EOF is first placed on the tape to mark the end for TAPASC so that the tape can be repositioned properly after the read.

The message "HOW MANY BACK ??" is then displayed and the two-digit response is input by GETDTA. The response is converted to binary and saved at OP93A to serve as a counter. The routine TPBKSP is executed until this counter goes to zero or a beginning of tape (BOT) indicator is found. When either of these events occurs, the routine branches to OP9111.

PROGRAM NAME - OP94

BRIEF DESCRIPTION - Rewinds the tape.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - REWIND

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine is self-explanatory.

PROGRAM NAME - OP95

BRIEF DESCRIPTION - Moves the tape forward over one record.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - TAPEIN

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine is self-explanatory.

PROGRAM NAME - OP96

BRIEF DESCRIPTION - Backspaces tape over one record.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - TPBKSP

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine is self-explanatory.

PROGRAM NAME - OP97, OP98

BRIEF DESCRIPTION - Requests that demand period data be printed/not printed on the Teletype at the end of each demand period.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - PRINTF

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

These routines are self-explanatory. (See PRINT routine for use of PRINTF.)

PROGRAM NAME - PRINT

BRIEF DESCRIPTION - Prints data on the Teletype for the demand period just ended.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - BCDASC, BCDAS1, PRIN9

DYNAMIC VARIABLES USED - CLKDAY, CLKHRS, CLKMIN, PRINX, PRINA, WLPUL, WLPUL+2, NINPS, PRINX1, NOUTS, TOTRCL, PRINX2

VARIABLES TO BE PRE-INITIALIZED - CLKDAY, CLKHRS, CLKMIN, NINPS, NOUTS

DETAILED DESCRIPTION -

This **routine** is invoked at the end of each demand period if PRINTF is set to a non-zero value. It is put on the wait list for 25 s at the end of the STATUS routine so that its execution will not interfere with any other freeze-time functions.

First the buffer BUF1 is filled with the current ASCII date/time stamp, and the next available buffer location is then stored at PRINX. A pointer into the table of input-data-block addresses is then saved at PRINX1 and the input counter is established at PRINA.

The relative address for the NPULL value in the data block is then loaded into the A register so that PRIN9 can compute the proper address within the data block. Once all inputs have been converted, the outputs are handled in a similar manner. The buffer is then terminated for output and the check pulse routine CHKPO is put back on the wait list for 25 s. The Teletype recall address is set to RINO and the buffer address is initialized to BUF1. Control then transfers to USE2 to start output to the Teletype.

PRIN9 is used to compute the first address within a specific data block to be output. This location is computed for the data block pointed to indirectly by PRINX1 and stored at PRINX2. If all inputs or outputs are done, then PRINA will be zero and control will transfer to PRIN9. Otherwise, a space is stored in the output buffer followed by four ASCII characters which represent the pulse count for the demand period just ended for the current input. The pointer at PRINX1 is then updated to point to the next data block. The loop continues at PRIN9 until all inputs or outputs are done and control transfers to PRIN99.

At PRIN99, a carriage return and a line feed are put at the end of the buffer and control returns to the user.

Note that little effort has been given to formatting the output. Therefore, with more inputs or outputs the display of the data on the Teletype may be difficult to read.

PROGRAM NAME - STATUS

BRIEF DESCRIPTION - Outputs demand data to the tape at the end of each demand period.

ON ENTRY - NA

ON EXIT - NA

SUBROUTINES CALLED - IDBUF, INPBUF, OUTBUF, USE1, XQTS

DYNAMIC VARIABLES USED - STATX, BUF2, INPTBL, INPPNT, WLPUL+2, WLPUL, NINPS, NOUTS, OUTTBL, PRINTF

VARIABLES TO BE PRE-INITIALIZED - INPTBL, NINPS, NOUTS, PRINTF

DETAILED DESCRIPTION -

The status routine gets control from the wait list at the end of each demand period. The first thing it does is to set the flag associated with the Form-A freeze-pulse relay. This opens the relay after it has been closed for 0.3 s. Next, a pointer into the output buffer BUF2 is established at STATX. The IDBUF routine is then called to place the date/time stamp in BUF2.

The inputs are now handled by initializing an input counter at STATA and a pointer into the input-data-block address table at INPPNT. The routines INPBUF and USE1 will then be called once for each input to initialize the buffer and update STATA and INPPNT. When all inputs are done, control transfers to STAT2 and a similar sequence is performed for the outputs.

At STAT4 the buffer is terminated with a $FF and the TAPOUT routine is put on the FIFO stack to await execution. If no printing of data is requested by PRINTF, then control transfers to RINO. Otherwise, the PRINT routine is put on the wait list for 25 s.

PROGRAM NAME - IDBUF

BRIEF DESCRIPTION - Places a date/time stamp in BUF2.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - CLKDAY, CLKHRS, CLKMIN, BUF2

VARIABLES TO BE PRE-INITIALIZED - CLKDAY, CLKHRS, CLKMIN

DETAILED DESCRIPTION -

Converts days, hours, and minutes to ASCII and saves the results in BUF2.

PROGRAM NAME - INPBUF

BRIEF DESCRIPTION - Gets the data from the input-data block pointed to by the X register and places it in BUF2.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - Address of input-data block

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA   B REGISTER - NA

X REGISTER - Address of next available location in output buffer

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - STATX

VARIABLES TO BE PRE-INITIALIZED - STATX

DETAILED DESCRIPTION -

On entry the X register points to the input-data block which is being queried. The last pulse time (LPULT), the maximum demand value (MAXDEM), the total pulse count (NPULT), and the pulses in the demand period just ended (NPULL) are stored on the stack. The output buffer pointer is then placed in the X register and the values are pulled from the stack in the reverse order from the way that they were placed on the stack. These values are stored in the output buffer and control returns to the calling program.

PROGRAM NAME - OUTBUF

BRIEF DESCRIPTION - Gets the data from the output-data block pointed to by the X register and places it in BUF2.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - Address of output-data block

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - Address of next available location in output buffer

STACK - Return address left on stack along with data bytes.

INTERRUPT STATUS - NA

EXIT MODE - Jump to INPBO

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

OUTBUF operates in a similar manner to INPBUF except that all values are related to an output-data block.

PROGRAM NAME - HEADER

BRIEF DESCRIPTION - Outputs header-type information about system parameters to tape.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Return address left on stack.

INTERRUPT STATUS - NA

EXIT MODE - JMP XQTS

SUBROUTINES CALLED - IDBUF, BINBCD, ICONST, USE1, OCONST, XQTS

DYNAMIC VARIABLES USED - BUF2, ID, NINPS, NOUTS, KWSCL, INPTBL, INPPNT, OUTTBL, STATA, KVASCL

VARIABLES TO BE PRE-INITIALIZED - ID, NINPS, NOUTS, KWSCL, INPTBL, OUTTBL, KVASCL

DETAILED DESCRIPTION -

This routine initializes BUF2 with a date/time stamp and initializes the output buffer pointer at HEADX. The station ID, the number of inputs and outputs, and the KW and KVAR scale factors are all fetched, converted to BCD if necessary, and stored in the output buffer.

Input- and output-table constant data are then fetched by ICONST and OCONST in a manner similar to INPBUF and OUTBUF of STATUS. A buffer terminator is then placed at the end of the buffer and the TAPOUT routine is placed on the FIFO stack. XQTS will return to the caller of HEADER.

PROGRAM NAME - ICONST

BRIEF DESCRIPTION - Puts input-data constants in buffer for output.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - Address of input-data block

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - Number of bytes on stack

X REGISTER - NA

STACK - Return address left on stack along with data bytes.

INTERRUPT STATUS - NA

EXIT MODE - JMP INPBO

SUBROUTINES CALLED - BINBCD

DYNAMIC VARIABLES USED - HEADX, ICONA

VARIABLES TO BE PRE-INITIALIZED - HEADX

DETAILED DESCRIPTION -

The X register points to the current input-data block on entry. The scale factor (INSCL), the power type and direction (PWRTYP and PWRDIR), and the maximum time between pulses (MPULT) are placed on the stack and terminated by a $EE byte on the stack. Then the output references are pushed onto the stack and the B register is incremented to count the number of outputs added to the stack. When all outputs are on the stack, control transfers to ICON2. A value is then pulled from the stack and the A register is cleared for the call to BINBCD, which returns the BCD values on the stack. The two most significant values are then ignored by two successive PULA instructions. The least significant value is then saved in the output buffer since all output references are at most two digits. When all output references are done, control transfers to INPBO to pull the rest of the values from the stack as specified by the number in the B register.

PROGRAM NAME - OCONST

BRIEF DESCRIPTION - Puts output-data constants in buffer for output.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - Address of output-data block

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - Number of bytes on stack

X REGISTER - NA

STACK - Return address left on stack along with data bytes.

INTERRUPT STATUS - NA

EXIT MODE - JMP INPBO

SUBROUTINES CALLED - BINBCD

DYNAMIC VARIABLES USED - HEADX, ICONA

VARIABLES TO BE PRE-INITIALIZED - HEADX

DETAILED DESCRIPTION -

The output scale (OUTSCL) and the power type (PTYPE) are placed on the stack and control transfers to INPB0 to pull the data from the stack and save them in the output buffer.

PROGRAM NAME - BCDAS2

BRIEF DESCRIPTION - Pulls a value from the stack to be processed by BCDAS1.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - Address of output data where results will be placed

STACK - *One data byte followed by the calling program return address*

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - Data byte to be converted B REGISTER - NA

X REGISTER - Same as on entry

STACK - Return address left on stack.

INTERRUPT STATUS - NA

EXIT MODE - JMP BCDAS1

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - BCDAA

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This is a utility program which pulls a value from the stack and then converts this value to ASCII and stores it in a buffer by branching to BCDAS1. Since this routine is called, the return address is on the top of the stack followed by the data to be converted. Therefore, the return address is removed from the stack and temporarily saved at BCDAA and in the B register. The value is then pulled from the stack and saved in the A register and the return address is then pushed back onto the stack.

PROGRAM NAME - EOF

BRIEF DESCRIPTION - Puts EOF on tape.

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - XQTS, SSP16, TAPCLO, REWIND, ACAR1

DYNAMIC VARIABLES USED - WLTOP+2

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

A $FF terminator is stored in the output buffer and the TAPOUT routine is placed on the FIFO stack to await execution. The status of the tape drive is then checked to make sure the tape has not reached the end-of-tape (EOT). If so, the message "**WORKING**" is displayed, the tape file is closed, and the tape is rewound. The message "PLEASE

REMOVE TAPE" is then displayed and the audible alarm is sounded. Control then returns to the calling program.

PROGRAM NAME - CRLF

BRIEF DESCRIPTION - Puts carriage return and line-feed characters in a specified buffer

ON ENTRY -

A REGISTER - NA B REGISTER - NA

X REGISTER - Address of buffer to put CRLF

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - Next available address in buffer

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine is self-explanatory.

PROGRAM NAME - SPAC1

BRIEF DESCRIPTION - Puts a space character in a specified buffer.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - Address of buffer where space is to be placed

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA   B REGISTER - NA

X REGISTER - Next available location in buffer

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - None

DYNAMIC VARIABLES USED - None

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

This routine is self-explanatory.

PROGRAM NAME - PASWRD

BRIEF DESCRIPTION - Requests a valid password from the operator.

ON ENTRY -

A REGISTER - NA   B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack on exit if valid password; otherwise, pulled from stack before exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS if valid password; otherwise, JMP RINO

SUBROUTINES CALLED - GETDTA, ASCBCD

DYNAMIC VARIABLES USED - CLKHRS, WLTOP+2

VARIABLES TO BE PRE-INITIALIZED - CLKHRS

DETAILED DESCRIPTION -

The message "PASSWORD=????" is displayed and a four-digit response is input by GETDTA. The first two digits of the response are converted to BCD and compared with CLKHRS. If a match is found, the second pair of digits is converted and checked. If all four digits match, the routine returns to the caller. If not, the return address is pulled from the stack and control transfers to RINO.

PROGRAM NAME - ALARM and ALAR1

BRIEF DESCRIPTION - Sounds alarm and may display error message.

ON ENTRY -

A REGISTER - Error number B REGISTER - NA

X REGISTER - NA

STACK - Return address of calling program

INTERRUPT STATUS - NA

ENTERED VIA - Subroutine call

ON EXIT -

A REGISTER - NA B REGISTER - NA

X REGISTER - NA

STACK - Return address pulled from stack on exit.

INTERRUPT STATUS - NA

EXIT MODE - RTS

SUBROUTINES CALLED - BCDASC, SSP

DYNAMIC VARIABLES USED - WLTOP+2, BUF5

VARIABLES TO BE PRE-INITIALIZED - None

DETAILED DESCRIPTION -

The ALARM entry point is entered if an error message is to be displayed. The A register equals the error message number which is displayed in the message "SYSTEM ERROR XX" where XX is the error message number.

ALAR1 then turns on the audible alarm and returns to the calling program.

DISTRIBUTION

DEFENSE DOCUMENTATION CENTER
CAMERON STATION, BUILDING 5
ALEXANDRIA, VA 22314
 ATTN DDC-TCA (12 COPIES)

COMMANDER
US ARMY MATERIEL DEVELOPMENT
 & READINESS COMMAND
5001 EISENHOWER AVENUE
ALEXANDRIA, VA 22333
 ATTN DRXAM-TL, HQ TECH LIBRARY

US DEPARTMENT OF COMMERCE
ASSISTANT SECRETARY FOR SCIENCE
 & TECHNOLOGY
WASHINGTON, DC 20230

US DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS
WASHINGTON, DC 20230

US ENERGY RESERACH & DEVELOPMENT
 ADMINISTRATION
ALBUQUERQUE OPERATIONS
P.O. BOX 5400
ALBUQUERQUE, NM 87115

US ENERGY RESEARCH & DEVELOPMENT
 ADMINISTRATION
TECHNICAL INFORMATION ORGANIZATION
P.O. BOX 62
OAK RIDGE, TN 37830

US DEPARTMENT OF THE INTERIOR
BONNEVILLE POWER ADMINISTRATION
P.O. BOX 3621
PORTLAND, OR 97208
 ATTN ROBERT ELLINGWOOD (10 COPIES)
 ATTN JAMES JONES

HARRY DIAMOND LABORATORIES
 ATTN LOWREY, AUSTIN, III, COL, COMMANDER/
  FLYER, I.N./LANDIS, P.E./
  SOMMER, H./OSWALD, R. B.
 ATTN CARTER, W.W., DR., TECHNICAL
  DIRECTOR/MARCUS, S.M.
 ATTN KIMMEL, S., PAO
 ATTN CHIEF, LAB 500
 ATTN RECORD COPY, BR 041
 ATTN HDL LIBRARY (5 COPIES)
 ATTN CHAIRMAN, EDITORIAL COMMITTEE
 ATTN TECH REPORTS, 013
 ATTN GIDEP OFFICE, 741
 ATTN LANHAM, C., 0021
 ATTN BUSCHER, D., (3 COPIES)
 ATTN RESSLER, M., (3 COPIES)